FPGA TABANLI SAYISAL SİNYAL İŞLEME ALGORİTMALARINA ÖZELLEŞTİRİLMİŞ YARDIMCI İŞLEMCİ TASARIMI

ABDULLAH GİRAY YAĞLIKÇI

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ

TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

AĞUSTOS 2014

ANKARA

Fen Bilimleri Enstitü onayı

_____

Prof. Dr. Ünver KAYNAK
Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

_____

Doç. Dr. Erdoğan Doğdu
Anabilim Dalı Başkanı

ABDULLAH GİRAY YAĞLIKÇI tarafından hazırlanan FPGA TABANLI SAYISAL SİNYAL İŞLEME ALGORİTMALARINA ÖZELLEŞTİRİLMİŞ YARDIMCI İŞLEMCİ TASARIMI adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

_____

Doç. Dr. Oğuz ERGİN
Tez Danışmanı

Tez Jüri Üyeleri

Başkan : .... _____

Üye : Doç. Dr. Oğuz ERGİN _____

Üye : .... _____

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada orijinal olmayan her türlü kaynağa eksiksiz atıf yapıldığını bildiririm.

Abdullah Giray Yağlıkçı

Üniversitesi : **TOBB Ekonomi ve Teknoloji Üniversitesi**
Enstitüsü : **Fen Bilimleri**
Anabilim Dalı : **Bilgisayar Mühendisliği**
Tez Danışmanı : **Doç. Dr. Oğuz ERGİN**
Tez Türü ve Tarihi : **Yüksek Lisans – Ağustos 2014**

Abdullah Giray Yağlıkçı

FPGA TABANLI SAYISAL SİNYAL İŞLEME ALGORİTMALARINA ÖZELLEŞTİRİLMİŞ YARDIMCI İŞLEMCİ TASARIMI

ÖZET

Sayısal sinyal işlemede yaygın olarak kullanılan fonksiyonların büyük bir veri seti üzerinde çalıştırılması durumunda paralelleştirilmesi, yürütme zamanını kritik bir şekilde azaltmaktadır. Farklı veriler üzerinde aynı işlemlerin tekrarlandığı algoritmalarda performans artışı sağlamak adına iş parçalarının paralel yürütülebilmesi için çok çekirdekli işlemciler, GPGPU, ASIC tasarımlar ve FPGA tabanlı sistemler algoritmanın koşturulacağı platformların başında gelir. Her bir platformun kendi avantajları ve dezavantajları olmakla beraber, düşük maliyet ile yüksek paralellik sağladığı için GPGPU ve FPGA'ler son yıllarda en yaygın kullanılan platformlardır. Bu tez, ASELSAN - TOBB ETÜ iş birliğinde yürütülen, çıktısı FPGA tabanlı ve OpenCL destekli, ölçeklenebilir ve özelleştirilebilir tasarıma sahip bir yardımcı işlemci ünitesi olan projenin donanım tasarımı kısmını kapsar. Tez çalışmalarına paralel olarak derleyici tasarımı yapılmış fakat tez içeriğine dahil edilmemiştir.

Anahtar Kelimeler: FPGA, hızlandırıcı, yardımcı işlemci, OpenCL.

University	:	**TOBB University of Economics and Technology**
Institute	:	**Institute of Natural and Applied Sciences**
Science Programme	:	**Computer Engineering**
Supervisor	:	**Assoc. Prof. Oğuz ERGİN**
Degree Awarded and Date	:	**M.Sc. – August 2014**

Abdullah Giray Yağlıkçı

TITLE OF THE THESIS

ABSTRACT

Typical digital signal processing algorithms executes the same DSP functions on different data sets. Parallelizing this process dramatically decreases execution time of such kind of functions. There are 4 popular platforms for parallelized applications: Many-core processors, GPGPUs, ASIC chips and FPGA based applications. Although each kind of platform has own pros and cons, GPGPU and FPGA based applications are more popular than others because of lower price and higher parallel processing capabilities. This MSc thesis consists of hardware design of a project which is managed by ASELSAN and TOBB ETÜ and the output of project is FPGA based OpenCL ready highly scalable and configurable co-processor. Although compiler works are in progress, this thesis only includes the harware design of co-processor.

Keywords: FPGA, accelerator, co-processor, OpenCL.

TEŞEKKÜR

Bu çalışmayı tamamlamamda emeği geçen değerli danışman hocam Doç. Dr. Oğuz Ergin'e; kıymetli çalışma arkadaşlarım Hasan Hassan, Hakkı Doğaner Sümerkan, Serdar Zafer Can, Serhat Gesoğlu, Volkan Keleş ve Osman Seçkin Şimşek'e; tez çalışmam sırasında beni destekleyen aileme ve değerli arkadaşlarım Fahrettin Koç, Tuna Çağlar Gümüş ve Emrah İşlek'e; projeye desteğinden ötürü ASELSAN'a ve çalışma ortamımızı sağladığı için TOBB ETÜ Mühendislik Fakültesi ve Fen Bilimleri Enstitüsüne teşekkür ederim.

İçindekiler

Şekil Listesi

Tablo Listesi

1. GİRİŞ

Sayısal sinyal işleme algoritmalarında sıklıkla aynı işlem, farklı veriler üzerinde uygulanmaktadır. Geleneksel işlemcilerde bu tarz bir uygulama her veri için işlemin peşpeşe tekrarlanması ile gerçeklenir. Oysa ki algoritmaların bu özelliği, farklı veriler için uygulanacak aynı işlemin sırayla değil paralel çalıştırılması ile kayda değer performans artışlarını beraberinde getirir. Örneğin N elemanlı iki vektörün skalar çarpımı, N adet çarpma işleminden ve ardından N adet verinin toplanmasından oluşur. N adet çarpma işleminden herhangi birinin bir diğerini beklemeye ihtiyacı yoktur. Bu çarpma işlemlerinin peşi sıra yapıldığı ve paralel yapıldığı durumlar karşılaştırıldığında, paralel olan yöntemde N kata yakın performans artışı gözlenir.Paralelleştirmenin azımsanamayacak performans avantajından dolayı paralel çalışmayı destekleyecek donanım tasarımları üzerinde pek çok çalışma yapılmıştır. Literatürde öne çıkan çalışmaları 4 başlık altında toplamak mümkündür.

Geleneksel işlemcilerde birden fazla iş parçacığının eş zamanlı çalıştırılabilmesi için çok çekirdekli mimari tasarımları yaygın olarak kullanılmaktadır. Çok çekirdekli işlemcilerde bir çekirdek üzerinde 1 veya daha fazla thread koşturulması ile sinyal işleme fonksiyonlarında paralellik sağlanmaktadır. Endüstriyel uygulamalarda kullanılan DSP(Digital Signal Processor) yongaları da çok çekirdekli işlemci mimarisine sahip özelleştirilmiş donanımlardır.[1] Bu tarz mimarilerde çekirdeklerin programlanabilir olması uygulamada esneklik sağlar. Genel amaçlı çok çekirdekli işlemciler, sinyal işleme uygulamalarında alternatiflerine göre daha az paralel ve daha yavaş kalırlarken DSP yongaları, ilave bir donanım olarak donanımın ömrünü kısaltmakta ve güncellenebilirliğini azaltmaktadır.[2]

Bilgisayar ekranına basılacak piksellerin renk ve parlaklık değerlerinin hızlı

ve paralel bir biçimde hesaplanabilmesi için geliştirilen grafik işlemcileri çok sayıda çekirdeğe sahiptir.[3] Hemen her bilgisayarda bulunan grafik işlemcilerinin genel amaçlı paralel hesaplama gerektiren işlerde kullanılması ekonomik ve yüksek performasnlı bir çözüm olarak kendini göstermiştir. Grafik işlemcilerinin genel amaçlı kullanımını destekleyen iki kutup olarak NVidia ve Khronos grubu, sırasıyla CUDA ve OpenCL desteği sağlayarak GPGPU (General Purpose Graphical Processor Unit) kullanımını yaygınlaştırmıştır. [4] [5] GPGPU programlama ile uygulamaların paralelleştirilmesi ek donanım gerektirmediği için ekonomik, çok sayıda çekirdekten oluşan donanımlar olduğu için yüksek derecede paralelleştirilebilir bir donanım alternatifidir. Ticari donanımlar olan grafik işlemcilerinin dezavantajı ise birinci önceliği piksel değeri hesaplayan çekirdeklerden oluşması ve çok özel amaçlı işlerde performans bakımından yetersiz kalmasıdır. Burada bahsi geçen yetersizlik buyruk kümesi tasarımı ile ilgilidir.

GPGPU ve DSP donanımlarının performans açısından yetersiz kaldığı durumlarda, donanım tasarımına müdahale edilebilen ASIC (Application Specific Integrated Circuit) tasarımlar ve FPGA(Field Programmable Gate Array) tabanlı sistemler ön plana çıkar. ASIC tasarımlar yarı iletken seviyesinde tasarlanan devrelerden oluşurken FPGA tabanlı sistemler, adından da anlaşılacağı üzere, FPGA yongalarında hazır bulunan LUT (Lookup Table), kapılar, bellekler vb. yapılar kullanılarak gerçeklenir. Her iki yaklaşımın diğerlerinden farkı yazılım seviyesinden donanım seviyesine inilmesi ile donanımın uygulamaya özelleştirilerek performans artışının sağlanmasıdır. ASIC - FPGA karşılaştırmasında ASIC uygulamalar daha alt seviyede, FPGA uygulamalar ise daha üst seviyede yapılır. Dolayısıyla ASIC tasarımdan alınan performans artışına FPGA seviyesinde erişilmesi mümkün değildir. Öte yandan ASIC uygulamaların, üretim gerektirdiği için maliyeti fazla, güncellenebilirliği azdır. [6]

Bu tez, sayısal sinyal işleme algoritmalarında yaygın olarak kullanılan fonksiyonların paralel çalıştırılması için tasarlanan FPGA tabanlı bir sistemin donanım tasarımını içerir. Söz konusu sistem ASELSAN ve TOBB ETÜ'nün ortak projesi olup, ASELSAN tarafından sayısal sinyal işleme uygulamalarında kullanılması planlanmaktadır. Dolayısıyla tasarımın temelini oluşturan kriterler ve fonksiyon listesi ASELSAN tarafından belirlenmiştir.

Tezin 2. bölümünde ASELSAN tarafından belirlenen tasarım kriterleri ve fonksiyon listesi özetlenmiş ve tasarım öncesi sistem özellikleri belirlenmiştir. 3. bölümde benzer özellikteki mimariler sunulmuş, avantajları ve dezavantajları tartışılmıştır. 4. bölümde buyruk kümesi ve boru hattı tasarımı anlatılmış, 5. bölümde ise mimari tasarımı alt modüllere ayrılarak her bir modülün tasarımı açıklanmıştır. 6. bölümde sonuçların sunumu ile tez sonlandırılmıştır.

2. GEREKSİNİM ANALİZİ

OpenCL ve CUDA altyapıları kullanılarak gerçeklenen sinyal işleme uygulamalarının, özelleştirilebilir, milli tasarım bir donanım üzerinde çalıştırılması amacı ile başlatılan projenin gerekinimleri 2.1 Proje Gereksinimleri başlığı altında sunulmuştur. **??** Paralelleştirmenin Başarıma Etkisi başlığı altında proje için performans metrikleri belirlenmiş, 2.3 Fonksiyonların Gerçeklenmesi başlığı altında, Tablo 2.1: Fonksiyon Listesi tablosunda verilen fonksiyonların matematiksel ifadeleri ve sayısal sistemler üzerinde gerçekleme algoritmaları sunulmuştur. Sunulan ifadeler **??** bölümünde kullanılacaktır.

2.1 Proje Gereksinimleri

Proje gereksinimleri şu şekildedir:

1. Tasarlanan işlemci çok çekirdekli mimariye sahip olmalıdır.
2. Tasarlanan işlemcinin buyruk kümesi OpenCL 1.2 desteklemelidir.
3. Tüm işlemler 32 bit integer ve floating point sayılar üzerinden yapılmalıdır. Floating point sayılar için IEEE754 standardı kullanılmalıdır.
4. Tasarım modüler olmalı alt modül sayıları parametrik tanımlanmalı, bütün mimari modülleri özelleştirilebilir olmalıdır.
5. Gelecek çalışmalarda tasarlanacak özel hesaplama ipcore modülleri için standart bir arayüzü desteklemelidir.
6. Tasarım sayısal sinyal işleme uygulamalarında sıklıkla kullanılan ve Tablo 2.1 içinde belirtilen fonksiyonları desteklemelidir.
7. Verilen bir matrisin kopyası oluşturulup kopya üzerinden işlem yapılmalıdır.
8. Reel sayılar matrisi oluşturulurken bellekte yalnızca reel sayıların sığabileceği bir alan kullanılmalıdır, karmaşık sayılar matrisi oluşturulurken reel ve imajiner kısımlar için ayrı yer ayrılmalıdır.
9. Satır, sütun veya alt matris üzerinde işlem yapılırken yalnızca ilgili veriler kopyalanmalıdır.

Tablo 2.1: Desteklenmesi beklenen fonksiyon listesi

Fonksiyon	**Açıklama**
Toplama	İki matrisin eleman eleman toplanması
	Matrisin tüm elemanlarına sabit eklenmesi
Çıkarma	İki matrisin eleman eleman farkı
	Matrisin tüm elemanlarından sabit çıkarılması
Çarpma	Matrislerin eleman - eleman çarpımı
	Matris çarpımı
	Matrisin tüm elemanlarının sabit ile çarpımı
Bölme	Matrislerin eleman - eleman bölümü
	Matrisin tüm elemanlarının sabite bölümü
Toplam	Matrisin satır toplamları
	Matrisin sütun toplamları
	Matrisin tüm elemanlarının toplamı
Max, Min,	Her satır için
Mean, Median	Her sütun için
	Matrisin tüm elemanları için
	En büyük elemanın ilk indisi
	Mutlak en büyük elemanın değeri
	Mutlak en büyük elemanın ilk indisi
Nokta çarpımı	İki vektörün nokta çarpımı
FFT/IFFT	Her satırın fourier ve ters fourier dönüşümü
	Her sütunun fourier ve ters fourier dönüşümü
Logaritma	Her eleman için doğal logaritma hesabı
	Her elemean için 10 tabanında logaritma hesabı
Eksponansiyel	10 tabanında eksponansiyel
	Doğal tabanda eksponansiyel
Büyüklüğü	Matrisin mutlak büyüklüğü
	Matrisin enerjisi
Evrişim	Dairesel konvolüsyon (Circular convolution)
	Doğrusal konvolüsyon (Linear convolution)
Eşlenik	Bir matrisin karmaşık eşleniği
Transpoz	Bir matrisin transpozu

Sonraki sayfada devam etmektedir.

Tablo 2.1 – devam

Fonksiyon	Açıklama
	Bir matrisin eşleniksiz transpozu
Determinant	Bir kare matrisin determinantı
Trigonometrik	Her eleman için sin/cos/tan değerleri
Filtreleme	Her satırı FIR ve IIR Filtreleme
	Her sütunu FIR ve IIR Filtreleme
Windowing	Hamming, Hanning ve Gaussian
Alt matris	Matrisin bir satırını al / değiştir
	Matrisin bir sütununu al / değiştir
	Matrisin bir alt matrisini al / değiştir
Türev	Bir vektörün 1. derecede türevi
Norm	Matrisin ve vektörün p. dereceden normu
Sıralama	Satır sıralama
	Sütun sıralama
	Matris sıralama (vektör sıralama gibi)
Varyans,	Satır bazlı
Standart	Sütun bazlı
Sapma	
	Matris bazlı
İşaret	Her bir eleman için signum fonksiyonu
Flip	Yatay ve düşey eksende flip
Karekök	Her eleman için karekök
Reverse	Elemanların sırasını tersine çevirir
Interpolasyon	Lineer interpolasyon
Karşılaştırma	Satır, sütun bazlı veya matris için karşılaştırma

Tasarlanan donanımın temel tasarım kararlarını oluşturan gereksinimler ve fonksiyon listesi incelenmiş, her bir matematiksel işlem için gerekli buyruklar ve donanım birimleri belirlenmiştir.

2.2 Paralelleştirmenin Başarıma Etkisi

Tablo 2.1 içinde belirtilen işlemlerin paralelleştirilmesi ile işlem sürelerinin kısalması beklenmektedir. Paralel hesaplamada işlem süresini belirleyen 4 unsur vardır.

Bunlardan birincisi bellek işlemlerine ayrılan süredir. Programlanabilir her sistemde olduğu gibi bir işlem veya işlem dizisi başlarken bellekten veri okunur, sonlandığında ise tekrar belleğe sonuçlar yazılır. İşlemler paralelleştirilse de paralelleştirilmese de bellek için harcanan süre toplamda yakındır. Citation Here Hem yazılım hem de donanım seviyesinde bellek işlemlerinde yerelliği artırmak bellek işlemlerinin daha hızlı işlenmesine olanak sağlar.

İkinci unsur paralelleştirmenin bir ölçüsü olan thread sayısıdır. Söz konusu işlem birbirinden bağımsız iş parçacıklarına bölünür ve her bir iş parçacığı farklı donanımlarda koşturularak paralel işleme sağlanır. Literatürde bu iş parçacıkları ingilizce ismi olan thread kelimesiyle ifade edilmekte ve thread kelimesinin buradaki anlamını taşıyan bir türkçe tercümesi bulunmamaktadır. Bu sebeple tezin devamında sürekli olarak thread kelimesi kullanılacaktır. Thread sayısındaki artış, programın daha paralel koşturulabilmesine olanak sağlar.

Üçüncü unsur donanımda gerçeklenmiş thread yolu sayısıdır. Her bir thread, bir thread yoluna atanır ve o yol üzerinde koşturulur. Eğer thread yolu sayısı thread sayısından büyük veya eşitse, tek seferde bütün threadler işlenir ve program sonlanır. Eğer thread sayısı, thread yolu sayısından fazla ise threadler, thread yolu sayısı kadar elemana sahip kümelere bölünür. NVidia'nın dokümanlarında warp ismi ile anılan bu thread kümelerinin her biri tek seferde işlenir. Toplam işlem süresi ise warp sayısına bağlı olarak artar. Thread yolu sayısının artırılması warp sayısında ve işlem süresinde azalmaya yol açar. Ancak fiziksel kısıtlardan dolayı thread yolu sayısının bir üst limiti vardır.

Dördüncü unsur ise her bir thread için harcanan yürütme zamanıdır. Thread başına düşen yürütme zamanı thread içindeki buyruk sayısına, buyrukların çevrim sayılarına, buyruklar arası veri bağımlılıklarına, işlemcinin boru hattı mimarisine ve işlemcinin frekansına bağlı olarak değişir.

Dolayısıyla bir paralelleştirilmiş bir uygulamanın yürütme zamanı denklem 2.1'de gösterildiği şekilde formüle dökülebilir.

$$t_{program} = t_{bellek} + t_{thread} x \frac{N_{thread}}{N_{threadyolu}} \& t_{thread} = N_{buyruk} x c_{ortalama} x T_{saat} \quad (2.1)$$

Burada $t_{program}$ program süresini, t_{bellek} bellek işlemleri süresini, t_{thread} thread süresini, N_{thread} toplam thread sayısını, $N_{threadyolu}$ toplam thread yolu sayısını, N_{buyruk} thread içindeki buyruk sayısını, $c_{ortalama}$ her buyruk için harcanan çevrim sayılarının ortalamasını, T_{saat} işlemci saatinin periyodunu ifade eder.

Thread yolu sayısının 1 olduğu durumda aynı anda tek bir thread işlenebilir. Dolayısıyla işlem paralelleştirilmemiş olur. Thread yolu sayısının sonsuza gitmesi halinde ise program süresi bellek işlemleri için harcanan zamana eşit olur.

Program süresi bileşenlerinin optimize edilmesi

Thread sayısı ve thread içindeki buyruk sayısı yazılım katmanında belirlenen değerlerdir. Bellek işlemleri için harcanan süre kaçınılmaz olmasına rağmen yazmaç öbeği, paylaşımlı bellek ve ana bellek ara yüzü gibi load ve store işlemleri ile ilgili donanımların tasarımlarında yapılan iyileştirmeler bellek için harcanan süreyi azaltabilir. Öte yandan işlemci frekansı ve işlemler için harcanan ortalama çevrim sayıları da hesaplama işlemlerinin süresini doğrudan belirleyen bileşenler olup optimize edilmesi gerekmektedir. Bu tarz bir optimizasyon için buyruk kümesi ve boru hattı mimarisi belirleyici yapılardır. Buyruk kümesi tasarımı için fonksiyon listesinde bulunan işlemler

2.3 Fonksiyonların Gerçeklenmesi

Fonksiyon listesinde belirtilen fonksiyonların tamamında veriler bellekten okunmakta ve sonuçlar yine belleğe yazılmaktadır. Dolayısıyla load ve store işlemleri fonksiyonlrın tümünde olmalıdır. Her bir fonksiyon için gerekli buyruklar ise her fonksiyonun kendi başlığı altında belirtilmiştir.

2.3.1 Toplama işlemi

İki matrisin eleman eleman toplamında her bir thread $C_{i,j} = A_{i,j} + B_{i,j}$ işlemini yapar. Bu işlem için ihtiyaç duyulan buyruklar floating point ve integer toplama buyruklarıdır. Bir matrisin sabit sayı ile toplanması durumunda ise her bir thread $C_{i,j} = A_{i,j} + k$ işlemini yapar. Burada k değeri integer veya floating point bir sayı olup, bellekten okunabileceği gibi anlık olarak da verilebilir. Dolayısıyla önceki buyruklara ek olarak integer ve float için anlık değer ile toplama buyrukları da gereklidir.

2.3.2 Çıkarma işlemi

İki matrisin eleman eleman toplamında her bir thread $C_{i,j} = A_{i,j} - B_{i,j}$ işlemini yapar. Bu işlem için ihtiyaç duyulan buyruklar floating point ve integer çıkarma buyruklarıdır. Bir matristen sabit sayının çıkarılması durumunda ise her bir thread $C_{i,j} = A_{i,j} - k$ işlemini yapar. Burada k değeri integer veya floating point bir sayı olup, bellekten okunabileceği gibi anlık olarak da verilebilir. Dolayısıyla önceki buyruklara ek olarak integer ve float için anlık değer çıkarma buyrukları da gereklidir.

2.3.3 Çarpma işlemi

MxN ve NxP büyüklükteki iki matrisin çarpılması işlemi MxP adet sonuç üretir. Bu sonuçların her biri için bir thread oluşturulur (toplamda MxP adet) ve her bir thread $C_{i,j} = \sum_{n=0}^{N}(A_{i,n} x B_{n,j})$ işlemini yapar. Bu işlem bir döngü içinde çarpma ve toplama yapılması ile gerçeklenir. Dolayısıyla döngü oluşturabilmek için gerekli atlama, karşılaştırma ve dallanma buyrukları gereklidir. Hesaplama için çarpma buyruğuna da ihtiyaç vardır. Bu işlemin gerçeklenmesinde performans artırmaya yönelik DSP uygulamalarında sıklıkla kullanılan çarp-topla (muladd) işlemi kullanılmalıdır.

Matrislerin eleman eleman çarpılması işleminde ise oluşturulan her bir thread $C_{i,j} = A_{i,j} x B_{i,j}$ işlemini yapar. Bu işlem için herhangi bir döngü yapısına ihtiyaç

kalmaksızın çarpma buyruğu yeterlidir.

Matrisin tüm elemanlarının sabit bir sayı ile çarpılması işleminde her bir thread $C_{i,j} = A_{i,j}/k$ işlemini yapar. Burada k sayısının anlık alınması istenirse anlık ile çarpma buyruğuna da ihtiyaç duyulur. Bütün çarpma ve çarp-topla buyruklarının float ve integer için versiyonlarının bulunması gerekir.

2.3.4 Bölme işlemi

İki matris arasında eleman-eleman bölme işlemi için oluşturulan her bir thread $C_{i,j} = A_{i,j}/B_{i,j}$ işlemini yapar. Bu işlem için float ve integer bölme buyrukları gereklidir. Bir matrisin sabit sayıya bölümü işleminde ise her bir thread $C_{i,j} = A_{i,j}/k$ işlemini yapar. Burada k sayısının anlık alınması istenirse anlık değere bölme buyruğunun gerçeklenmesi gerekir.

2.3.5 Toplam işlemi

Bir matrisin satır toplamlarını, sütun toplamlarını veya tüm elemanların toplamını bulur. Bütün program ikişerli eleman toplamlarından oluşur. Örneğin tüm satır toplamları için satır başına log_2N kez, sütun toplamları için sütun başına log_2M kez ardışık toplama işlemi yapılması gerekir. Tüm elemanların toplamı içinse $log_2(MxN)$ kez ardışık toplama işlemi yapmak gerekir. İhtiyaç duyulan buyruk ise toplama buyruğudur.

2.3.6 Max,Min,Ortalama,Ortanca, Karşılaştırma

Verilen herhangi N elemanlı bir veri seti üzerinde (matris veya matrisin bir parçası) max ve min hesapları için ardışık log_2N adet karşılaştırma işlemi yapılır. Ortalama hesabı için elemanların toplamı bulunup bölme işlemi yapılır. Ortanca hesabı için ise sıralama yapılması gerekmektedir. Merge-sort algoritması düşünülürse, log_2N ardışık karşılaştırma ile sıralama yapılır ve ortanca terim bulunur. Bu fonksiyonlar için öncekilerden farklı olarak karşılaştırma buyrukları gereklidir.

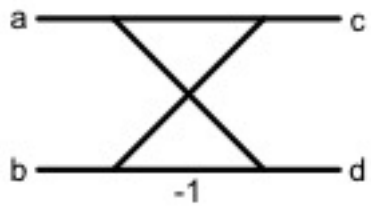
2.3.7 Nokta çarpımı

v_1 ve v_2 iki adet N elemanlı vektör olsun $v_1.v_2 = \sum_{i=1}^{N} v_1[i]xv_2[i]$ şeklinde tanımlıdır. Daha önce matris çarpımında belirtildiği şekilde çarp, çarp-topla ve topla buyrukları kullanılarak bu işlem gerçekleştirilir. Burada her bir çarpımı oluşturmak için ayrı bir thread oluşturularak paralellik sağlanabilir.

2.3.8 FFT/IFFT

Ayrık zamanda fourier ve ters fourier dönüşümü için günümüzde yaygın olarak kullanılan algoritma Cooley-Tukey FFT algoritmasıdır. ref Bu algoritmanın radix-2 decimation in time gerçeklemesinin uygulanması durumunda her bir thread bir butterfly işlemini çalıştırır. 8 elemanlı bir vektörün FFT işlemi Şekil 2.3.8'de sunulmuştur.

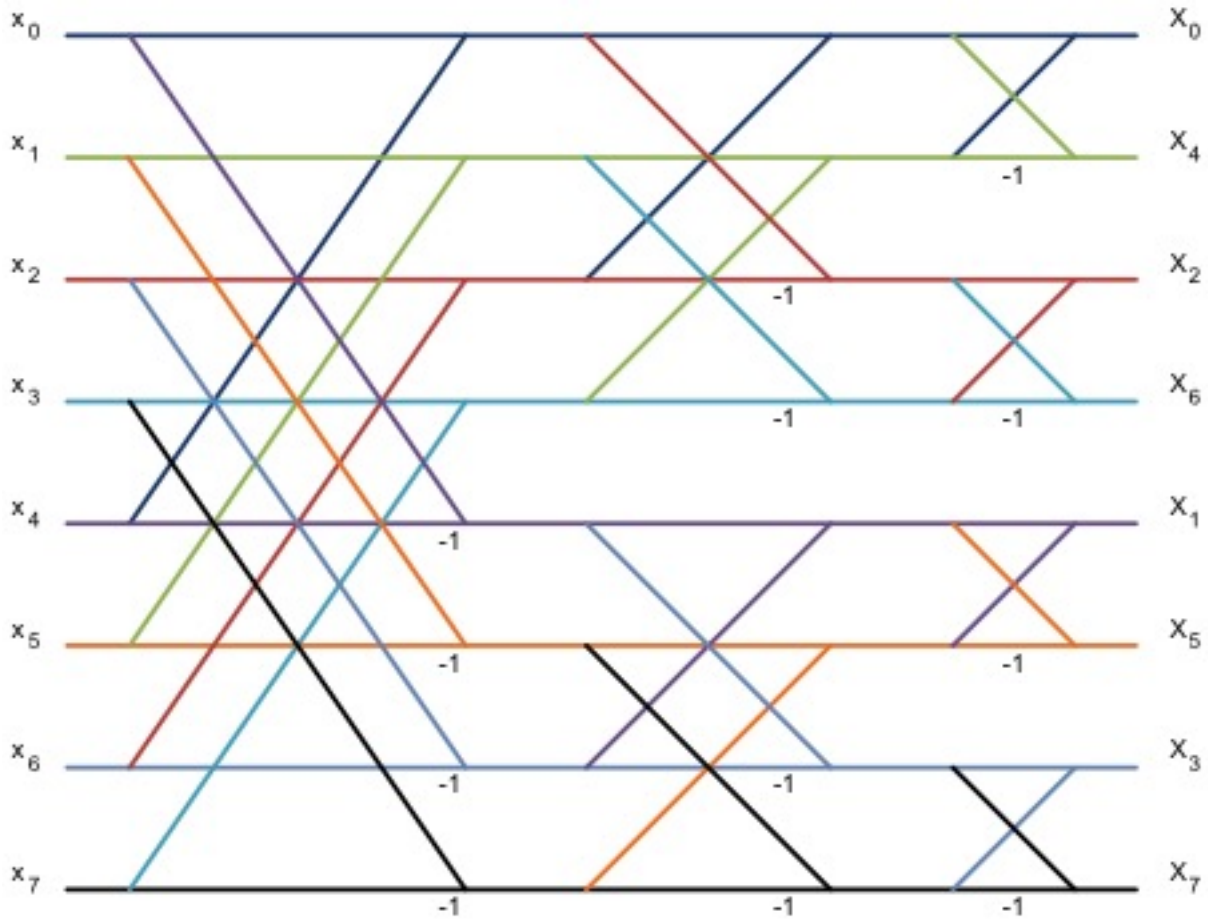
Fourier transformu alınacak olan giriş sinyali x(n), bu sinyalin fourier transformu ise X(n) olsun. Radix-2 yönteminde x(n) vektörünün elemanları tek indisli elemanlar ve çift indisli elemanlar olarak ayrılıp, ikişerli gruplara bölünürler. Daha sonra her bir eleman kendinden N/2 uzaktaki eleman ile butterfly işlemine alınır. Şekil 2.3.8'de sunulan algoritma, şekil 2.3.8'de çizimi sunulan butterfly işlemlerinden oluşur. Her bir butterfly işleminde yapılan hesaplama denklem **??**'de gösterildiği gibidir.



Şekil 2.1: Radix 2 için butterfly işlemi

$k_1 = cos(-\frac{2pi}{N})\&k_2 = sin(-\frac{2pi}{N})$

$$\begin{bmatrix} c_{Re} & c_{Im} \\ d_{Re} & d_{Im} \end{bmatrix} = \begin{bmatrix} a_{Re} & a_{Im} \\ a_{Re} & a_{Im} \end{bmatrix} + \begin{bmatrix} b_{Re} & b_{Re} \\ -b_{Re} & -b_{Re} \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} + \begin{bmatrix} -b_{Im} & b_{Im} \\ b_{Im} & -b_{Im} \end{bmatrix} \begin{bmatrix} k_2 \\ k_1 \end{bmatrix} \quad (2.2)$$



Şekil 2.2: 8 noktalı sinyal için FFT Radix 2 algoritması

Denklem 2.2'de görüldüğü üzere her bir butterfly işlemi matris çarpımları ve matris toplamları şeklinde ifade edilebilir. İşleme alınan parametreler a ve b sayılarının reel ve imajiner kısımlarının yanı sıra $sin(-2\pi/N)$ ve $cos(-2\pi/N)$ değerleridir. Burada N değeri sonuç vektörünün her bir elemanın indisi olup, bir eleman için bir kez hesaplanır.

FFT gerçeklemesi için sin ve cos değerlerinin hesaplanabilmesi gerekmektedir. Dolayısıyla matris çarpma ve toplama işlemlerinin yanı sıra trigonometri buyrukları da gerekmektedir.

2.3.9 Logaritma

Verilen bir veri setinin her elemanı için doğal logaritma (e tabanında) ve 10 tabanında logaritma hesaplanması gerekir. Xilinx tarafından sağlanan IPCore ile doğal logaritma hızlı bir şekilde hesaplanabilmektedir. $log_a(x) = log_e(x)/log_e(a)$ denkliğinden faydalanılarak herhangi tabanda logaritma hesaplanabilir. Burada buyruk kümesine $log_e x$ buyruğunun da eklenmesi gerekir.

2.3.10 Eksponansiyel

Verilen bir veri setinin her elemanı için 10^x ve e^x değerlerinin hesaplanması gerekir. Xilinx tarafından sağlanan IPCore ile e^x hızlı bir şekilde hesaplanabilmektedir. $a^b = e^{bxlog_e a]}$ denkliğinden faydalanılarak herhangi a^x değeri hesaplanabilir.

2.3.11 Norm

Sinyal işlemede yaygınlıkla kullanılan matris normları 1, 2 ve ∞ normlardır. 1-norm sütun toplamlarının maksimumu şeklinde tanımlıdır. $\|X\|_1 = max_j(\sum_i(a_{ij}))$ 2-norm matrisin karesinin en büyük özdeğerinin karekökü olarak tanımlanmıştır. $\|X\|_2 = \sqrt[2]{max(eig(AxA))}$. Bir matrisin ∞ normu ise satır toplamlarının maksimumu olarak tanımlanmıştır. $\|X\|_\infty = max_i(\sum_j(a_{ij}))$.[7]

2-norm için kullanılacak özdeğerlerin hesaplanması bu işlemin bir alt parçasıdır. Özdeğer hesaplama algoritmasının gerçeklenmesinde matris büyüklüğü sabit kabul edilemeyeceği ve toplama ve kaydırma gibi temel işlemler cinsinden paralelleştirilebilir bir program yazılabileceği için özdeğer hesaplama işini yazılım seviyesinde gerçeklemek daha uygundur.[8]

2.3.12 Evrişim

Evrişim (ing. convolution) sinyal işlemede sıklıkla kullanılan bir işlemdir. İki vektörün evrişimi $Conv(f,g)[n] = \sum_{m=-\infty}^{\infty}(f[n]xg[n-m])$ şeklinde hesaplanır. Formülden de anlaşılacağı üzere evrişim sonuç vektörünün her bir elemanı bir

dizi çarpımın toplamı şeklinde hesaplanır. Burada sonuç vektörünün her bir elemanı için ayrı thread koşturulursa, 1 çarp ve N-1 çarp-topla buyruğu ile sonuç hesaplanmış olur.

2.3.13 Alt Matris, Flip, Reverse, Eşlenik ve Transpoz

Karmaşık sayılar düzleminde a + ib şeklinde tanımlanan bir karmaşık sayının eşleniği a - ib sayısıdır. Sayısal sistemlerde karmaşık bir sayının reel ve imajiner kısımları ayrı değerler olarak tutulduğundan imajiner kısmın işaretinin değiştirilmesi eşlenik hesaplaması için yeterlidir. Transpoz işlemi ise matris elemanlarının yerlerinin değiştirilmesi yani okunup işlem yapılmadan yazılması ile gerçeklenir. Alt matris, flip ve reverse işlemleri ise yalnızca okuma ve yazma bellek işlemlerinden oluşur.

2.3.14 Determinant

Genel geçer determinant hesaplama yönteminde matris, 2x2 boyutunda alt parçalarına ayrılır determinantlarından yeni bir matris oluşturulur, oluşan matris üzerinde yine aynı işlem uygulanır. En son tek elemana düştüğünde matrisin determinantı hesaplanmış olur. 2x2 matrisin determinantı $det(A) = a_{00}xa_{11} + a_{01}xa_{10}$ şeklinde hesaplanır. Bu işlem 1 çarpma 1 çarp-topla buyruğu ile gerçeklenebilir.

2.3.15 Trigonometrik İşlemler

Tüm trigonometrik işlemler sin ve cos cinsinden ifade edilebilir. FPGA platformunda Xilinx IPCore kullanılarak sin ve cos işlemleri hızlıca hesaplanabilir.

2.3.16 Filtreleme ve Windowing

Filtreleme ve windowing işleminde önceden belirlenmiş bir vektör veya matris işleme alınacak vektör yada matris üzerinde gezdirilerek eleman eleman çarpma ve

toplama işlemleri yapılır. Gereksinimlerde belirtilen Hamming Hanning Gaussian windowing işlemlerinde window değişir, işlem aynıdır. FIR ve IIR filtrede de temel işlemler windowing ile aynı olup, algoritma seviyesinde farklılıklar ile gerçeklenir. Bir f vektörü üzerine uygulanacak g maskesi ile filtreleme veya windowing $y[n] = \sum_{i=0}^{N} f[i]xg[i]$ şeklinde gösterilebilir.

2.3.17 Türev

Bir vektörün türevi, ayrık zamanda ardışık elemanların farkı şeklinde tanımlıdır. N elemanlı bir vektörün türevinin hesaplanması için N adet thread oluşturulur ve her bir thread bir çıkarma işlemi yapar.

2.3.18 Sıralama

Satır, sütun ve matris elemanlarının sıralanması uygulaması herhangi bir sıralama algoritması ile gerçeklenebilir. Alt seviyede her bir thread basit karşılaştırma işlemleri yapar.

2.3.19 Varyans ve Standart Sapma

Varyans ve standart sapma için dizinin ortalaması hesaplanır, elemanların ortalamaya uzaklıkları üzerinden toplama, karesini alma ve karekök alma gibi işlemler yapılır.

2.3.20 Karekök

Karekök işlemi kendi başına bir uygulama olarak değil diğer uygulamaların içinde bir işlem olarak kendini gösterir. Xilinx IPCore kullanılarak karekök işlemi hızlı bir şekilde yapılabildiğinden IPCore kullanımı tercih edilmiştir.

2.3.21 İşaret

İşaret fonksiyonu bir matris veya vektörün tüm elemanları için eleman pozitif ise 1, 0 ise 0, negatif ise -1 değerini döndürür. Eleman sayısı adetinde thread oluşturularak hızlı bir şekilde bu işlem gerçekleştirilebilir.

2.3.22 Interpolasyon

Interpolasyon işlemi, ardışık elemanların ağırlıklı ortalamalalarının hesaplanması ile gerçeklenir. Temel toplama, çarpma, kaydırma, bölme gibi işlemler ile ağırlıklı ortalama hesaplanır. Eleman sayısı kadar thread oluşturularak işlem paralelleştirilebilir.

2.3.23 Özet

Listedeki fonksiyonların incelenmesi ile gerekli hesaplama buyrukları çıkarılmıştır. Fonksiyon listesinin gerçeklenebilmesi için gerekli buyruklar Tablo 2.2'de sunulmuştur.

Tablo 2.2: Gerekli Hesaplama Buyrukları

Fonksiyon	**Açıklama**
add, addi, fadd	Float ve tamsayı değerleri için toplama ve tamsayı için anlık toplama işlemleri
sub, subi, fsub	Float ve tamsayı değerleri için çıkarma ve tamsayı için anlık çıkarma işlemleri
mul, muli, fmul	Float ve tamsayı değerleri için çarpma ve tamsayı için anlık çarpma işlemleri
div, divi, fdiv	Float ve tamsayı değerleri için bölme ve tamsayı için anlık bölme işlemleri
fma, ffma	Float ve tamsayı değerler için fused multiply add işlemi

Sonraki sayfada devam etmektedir.

Tablo 2.2 – devam

Buyruk	Detay
sin, cos, fsin, fcos	Float ve tamsayı değerler için trigonometrik işlemler
log, flog	Float ve tamsayı değerler için e tabanında logaritma işlemi
exp, fexp	Float ve tamsayı değerler için e^x işlemi
shl, shr, shra	Aritmetik ve mantık kaydırma buyrukları
sqrt, fsqrt	Float ve tamsayı değerler için karekök işlemi
cmp, br, jump	Döngü ve koşul oluşturabilmek için gerekli karşılaştırma, dallanma ve atlama buyrukları

3. BENZER MİMARİLER VE ÖNCEKİ ÇALIŞMALAR

Paralel hesaplama için literatürde var olan mimariler Flynn taksonomisi adıyla binilen bir sınıflandırmaya tabidir. Söz konusu donanım, özelliklerine göre bu sınıflandırmada bir sınıfa yerleştirilir. Literatür taramasında öncelikle bu sınıflandırmadan bahsedilmiş, ardından belirlenen sınıfta ön plana çıkan mimariler incelenmiştir.

3.1 Paralel işleme taksonomisi

Bilgisayar bilimlerindeki tüm uygulamalar ve donanımlar paralellik bakımından 4 sınıfta incelenir. Bu sınıflandırma literatürde Flynn Taksonomisi adıyla geçer [9]. Literatürdeki kısaltmalarıyla bu 4 sınıf, SISD (Single Instruction Single Data), SIMD (Single Instruction Multiple Data), MISD (Multiple Instruction Single Data) ve MIMD (Multiple Instruction Multiple Data) şeklinde isimlendirilir.

width=10cm

Şekil 3.1: Flynn Taksonomisi

SISD mimarilerde herhangi bir paralellikten bahsetmek söz konusu değildir. Tek thread çalıştıran mimariler SISD için örnek olarak gösterilebilir.

SIMD mimariler bir buyruğun birden fazla veri seti üzerinde çalıştırıldığı mimarilerdir. Örneğin NxM büyüklüğünde matrislerin toplandığı bir matris toplama işleminde NxM adet veri seti üzerinde basit bir toplama işlemi yapılmaktadır. Gereksinimler ışığında SIMD mimari bu çalışmanın mimari alternatifleri

arasındadır.

MISD mimariler bir veri seti üzerinde birden fazla buyruğun çalıştırıldığı mimarilerdir. MISD yaygın olarak hata düzelten sistemlerde tercih edilir. Örneğin uzay ortamında çalışması hedeflenen bir hesaplama biriminin ışımalara maruz kalması sebebiyle hesaplamasında veya kaydettiği sonuçlarda yanlışlık olabilir [10]. Bu tarz potansiyel problemlere önlem olarak yapılan her işlem aynı veri seti üzerinde birden fazla kez yapılır ve sonuçlar birden fazla yerde saklanır. Daha sonra aynı verinin kopyaları arasında karşılaştırma yapılırak hatalar algılanır ve düzeltilir.

MIMD mimariler bu taksonominin en karmaşık mimarileri olup birden fazla veri seti üzerinde birden fazla buyruğun çalıştırıldığı mimarilerdir. Buna örnek olarak günümüzde kullanılan CPU mimarileri verilebilir. Örneğin Intel Larrabee mimarisi GPU mimarisinde işlevsellik bakımından geliştirilmiş çekirdeklerin kullanılması ile ortaya çıkan bir GPGPU (General Purpose Graphical Processing Unit) olup aynı anda birden fazla veri seti üzerinde birden fazla işlemi koşturabilmektedir [11].

Proje gereksinimlerinde ve fonksiyon listesinde belirtilen, hedef donanım hakkındaki ihtiyaçlar, Flynn taksonomisinde SIMD sınıfı ile örtüşmektedir. MIMD bir mimari ise proje gereksinimlerinin üzerinde bir özellik olup, eniyileştirmeye yönelik bir çalışma olabilir.

3.2 Mevcut Mimariler

Gereksinimlerde belirtilen fonksiyonlar ışığında hesaplamalar için kullanılacak modüller belli IPCore donanımları ve basit hesaplama modüllerinden oluşur. Paralel işlemeye özel donanımlarda yürütme zamanının en büyük bileşeni verilerin okunması ve yazılmasından oluşan bellek işlemleri olduğu için mimari seviyesinde donanım özelliklerini belirleyici unsur, veri yolu tasarımıdır.

Veri yolu mimarisi, bellek, yazmaç öbekleri ve hesaplama birimleri arasındaki bağlantı ile bu yapıların mimari hiyerarşisinden oluşur. Literatürde öne çıkan veri yolu mimarileri üç sınıfta değerlendirilebilir: Homojen az çekirdekli işlemciler,

homojen çok çekirdekli işlemciler ve heterojen yapıdaki işlemciler.

3.2.1 Homojen az çekirdekli işlemciler

Homojen az çekirdekli mimariler birbirinin aynı olan az sayıda yüksek işlem kapasiteli çekirdeklerin 2. veya daha üst seviyede önbellekler üzerinden veri paylaşımı sağladığı işlemcilerdir. Bu mimaride her işlemci çekirdeğin kendisine ait bir önbelleği vardır. Bunlar bir interconnect yardımıyla bütünleşik bir paylaşımlı önbelleğe bağlanırlar. Bu yapıya örnek olarak Intel'in Nehalem işlemcisi gösterilebilir [12] [13]. Nehalem mimarisinde hususi önbellek 2 seviyeye ayrılmıştır ve paylaşımlı önbellek 3. seviyeyi oluşturmaktadır. Çekirdekler 3. seviye önbelleğin ardından Şekil 3.2.1'deki gibi bir bellek denetleyicisi ile sistemin ana belleğine bağlanır.

width=50

Şekil 3.2: Nehalem

3.2.2 Homojen çok çekirdekli işlemciler

Homojen çok çekirdekli mimariler birbirinin aynı olan çok sayıda düşük işlem kapasiteli çekirdeklerden oluşan yapılardır. Bunlara örnek olarak grafik işlemcileri verilebilir [14]. Şekil 3.2.2'teki gibi bir yapıya sahip olan grafik işlemcilerde amaç, paralelliği ön plana çıkarmak, çok sayıda verinin aynı anda işlenebilmesine olanak sağlamaktır. Az çekirdekli işlemcilerin aksine belleği kullanmak isteyen daha çok çekirdek olacağından bu mimarilerde bellek açısından bir darboğaz oluşmasına sebep olur. Homojen çok çekirdekli mimarilerin bellek hiyerarşisi 2 seviyeli önbellek ve ana bellekten oluşur. Her iki önbellek de çekirdek adacığında paylaşımlıdır. Az çekirdekli mimarilerin aksine çok çekirdekli mimarilerde genel bir yazmaç öbeği tüm çekirdeklerin erişimine açık olup yürütme zamanında her bir çekirdeğe özel olarak atanır.

width=300pt

Şekil 3.3: Nvidia GPU

Homojen az çekirdekli mimariler genel amaçlı kullanılan CPU (Central Processing Unit) mimarilerinde tercih edilirken çok çekirdekli mimariler GPU (Graphical Processing Unit) ön plana çıkar. CPU çekirdekleri yüksek işlem gücüne sahip ve az sayıda iken GPU çekirdekleri düşük işlem gücüne sahip ve çok sayıdadır. CPU üzerinde koşturulan programların dallanma ve bellek işlemleri için harcadığı zamanın azaltılması için çekirdeklere yakın büyük kapasiteli önbellekler kullanılır. GPU çekirdeklerinin sayıca fazla olması paralel hesaplamayı ön plana çıkarmakta ve ana bellek erişimi için kullanılan veri yolu genişliği, önbellek büyüklüğünden daha önemli bir kriter olmaktadır. Tablo 3.1 içinde CPU ve GPU mimarilerinin bellek özellikleri sunulmuştur [15].

Tablo 3.1: CPU GPU Bellek Karşılaştırması

	CPU	**GPU**
Bellek	6 - 64 GB	768 MB - 6 GB
Bellek Bant Genişliği	24 - 32 GB/s	100 - 200 GB/s
L2 Önbellek	8 - 15 MB	512 - 768 KB
L1 Önbellek	256 - 512 KB	16 - 48 KB

Homojen çok çekirdekli mimarilere verilebilecek bir örnek de sunucu sistemlerinde kullanılan Tile mimarisidir. [16] Bu mimaride 36-100 arasında RISC işlemciden oluşan çekirdekler birbirlerine bağlanarak yüksek paralellik elde edilir. Tile mimarisinde bellek mimarisi olarak şekil 3.2.2'te sunulan NUCA (non-uniform cache architecture) önbellek mimarisi kullanılır. Bu mimaride çekirdeklerin her birinin kendine ait özel önbelleği vardır. İkinci seviye önbellek olarak diğer çekirdeklerin önbellekleri kullanılır. Örnek olarak, 64 çekirdekli bir işlemcide her bir çekirdeğin 32 KB önbelleği olduğunu varsayarsak; 1 numaralı çekirdeğin 32 KB 1. seviye ve 2016 KB 2. seviye önbelleği olacaktır. Bu tasarımda herhangi bir çekirdeğin diğer tüm çekirdeklerin önbelleklerine bağlantısı olmalıdır. Çekirdek sayısının artması ile bu gereksinim bir wiring problemine dönüşür ve uzun yollar kritik yolu etkileyerek toplam gecikmeye katkıda bulunabilir. Bu kısıttan dolayı Tile mimarisinde 2 boyutlu bir MESH ağı kurulmuş ve her bir çekirdek bu ağdaki bir node olarak yerleştirilmiştir. Her node bir çekirdek, bir önbellek ve bir routerdan oluşur. Bir çekirdek kendinden farklı tüm çekirdeklerin ön belleklerini ikinci seviye ön bellek olarak kullandığından MESH network

üzerinden her birine erişimi vardır. Ancak fiziksel olarak kendisine uzak olan veriye erişebilmesi komşuluğundaki routerlar üzerinden her seferinde bir birim şeklindedir. Bu davranış satranç tahtası üzerinde şahın hareketi gibi düşünülebilir. MESH network yapısında tüm verilere erişim hızı aynı olmamakla birlikte, maksimum gecikme, node sayısının karekökü ile orantılı olarak artar.

width=10cm

Şekil 3.4: Tile Mimarisi

3.2.3 Heterojen yapıdaki işlemciler

Birbirinin aynı olan çekirdeklerin az veya çok sayıda gerçeklenmesi ile elde edilen paralel hesaplama donanımları çoğu uygulamada performans açısından yeterli gelse de, bir takım uygulamalarda sık kullanılan bazı işlemlerin hızlandırılması adına özel donanımlar gerçeklenir. Literatürde bu tip işlemciler heterojen yapıdaki işlemciler olarak adlandırılır. Heterojen mimariler doğrudan amaca yönelik hazırlandıkları için çok farklı mimari yapılarda gerçeklenebilirler. Heterojen mimarilerin temel özelliği bir işi her zaman o işi en hızlı yapan donanıma vermeleridir. Bu sebeple sık kullanılan hemen her işlem için ayrı hesaplama birimleri yerleştirilerek, özel fonksiyonların yazılım seviyesinden donanım seviyesine indirilmesi sağlanır. Örnek olarak şekil 3.2.3'te sunulan heterojen mimari çizimi Playstation oyun konsollarında kullanılan Cell mimarisine aittir.

width=10cm

Şekil 3.5: Playstation Cell Mimarisi

Şekil 3.2.3'te gösterilen Cell mimarisinde PPE (Power processing element) ana işlemci olup, SPE (Synergistic processing element) bloklarının her biri ise DSP benzeri SIMD işlemcilerdir.

4. GENEL İŞLEMCİ MİMARİSİ

İşlemci tasarımı buyruk kümesinin tasarlanması ile başlar. Daha sonra buyrukların koşturulabilmesi için gerekli donanımlar belirlenir ve bu donanımların yüksek verimli kullanımını sağlamak için boru hattı mimarisi tasarlanır. Tezin bu bölümünde öncelikli olarak buyruk kümesi mimarisi anlatılacak, ardından her buyruğun ihtiyaç duyduğu hesaplama modülleri belirlenecek, sonrasında kullanım senaryoları üzerinden boru hattı mimarisi tasarımı anlatılacaktır. Son olarak Tosun işlemcisinin veri yolu mimari yapısı ve tasarım kararları üzerinde durulacaktır.

Literatür özetinde belirtilen mimari alternatifleri, çekirdek sayısı ve homojen - heterojen çekirdekler bakımından farklı sınıflara ayrılmıştır. Hedeflenen donanım bir FPGA platformudur. FPGA platformları, ASIC tasarımlara göre daha düşük saat sıklığında çalışabildiğinden, uygulamanın yüksek seviyede paralelleştirilmesi ile faydalı bir ürün oluşturulabilir.

4.1 Buyruk Kümesi Mimarisi

Hedeflenen işlemciye benzer özelliklerde mevcut paralel işlemcilerin buyruk kümesi mimarileri incelenmiş, gereksinim analizinde fonksiyonların gerçeklenebilmesi için gerekli olarak belirlenen buyruklar bu buyruk kümesi mimarilerine eklenerek Tosun işlemcisi için bir buyruk kümesi mimarisi oluşturulmuştur. Mevcut buyruk kümelerinin incelenmesinin sebebi paralel işlemcilerin mimari özelliklerinden bağımsız olarak sahip olması gereken ortak özelliklerin bulunmasıdır. Bu özelliklerden bazıları yükleme ve saklama operasyonları, threadler arası senkronizasyonun sağlanması, çekirdeklerin bellek erişimlerinde kullanılan adres

hesaplamaları, yazmaçlar üzerinde yapılan okuma ve yazma işlemleridir.

Tosun buyruk kümesi mimarisinin oluşturulmasında NVidia PTX [17] buyruk kümesi paralel işleme mimarisi olarak temel alınmıştır. Ayrıca adres hesapları, dallanmalar, temel aritmetik ve mantık işlemleri gibi her işlemcinin sahip olması gereken temel buyruklar için de Intel x86 [18] ve MIPS [19] buyruk kümeleri referans alınmıştır.

Tosun buyruk kümesi mimarisinde bulunmasına karar verilen buyruklar tablo 4.1 içinde sunulmuştur.

Tablo 4.1: Tosun Buyruk Listesi

Buyruk	**Açıklama**	**Türü**
addi	$r_d = r_{s1}$+anlık	Anlık
andi	$r_d = r_{s1}$&anlık	Anlık
ori	$r_d = r_{s1}$|anlık	Anlık
xori	$r_d = r_{s1} \oplus$anlık	Anlık
divi	$r_d = r_{s1}$/anlık	Anlık
muli	$r_d = r_{s1} x$anlık	Anlık
subi	$r_d = r_{s1}-$anlık	Anlık
movi	$r_d(altyar ß s ß)$ =anlık	Anlık
movhi	$r_d(styar ß s ß)$ =anlık	Anlık
fabs	$r_d = \|r_{s1}\|$	Y1
fadd	$r_d = r_{s1} + r_{s2}$	Y2
fcom	$r_d = com(r_{s1}, r_{s2})$	Karşılaştırma
fdiv	$r_d = r_{s1}/r_{s2}$	Y2
fmul	$r_d = r_{s1} x r_{s2}$	Y2
fsqrt	$r_d = sqrt(r_{s1})$	Y1
fcos	$r_d = cos(r_{s1})$	Y1
fsin	$r_d = sin(r_{s1})$	Y1
ffma	$r_d = r_{s1} x r_{s2} + r_{s3}$	Y3
ffms	$r_d = r_{s1} x r_{s2} - r_{s3}$	Y3
fmin	$r_d = min(r_{s1}, r_{s2})$	Y2
fmax	$r_d = max(r_{s1}, r_{s2})$	Y2
fln	$r_d = log_e(r_{s1})$	Y1

Sonraki sayfada devam etmektedir.

Tablo 4.1 – devam

Buyruk	Açıklama	Türü
fmod	$r_d = r_{s1}\%r_{s2}$	Y2
f2int	$r_d = r_{s1}$	Y1
int2f	$r_d = r_{s1}$	Y1
fchs	$r_d = -r_{s1}$	Y1
fexp	$r_d = e^{r_{s1}}$	Y1
add	$r_d = r_{s1} + r_{s2}$	Y2
and	$r_d = r_{s1}\&r_{s2}$	Y2
or	$r_d = r_{s1}\|r_{s2}$	Y2
xor	$r_d = r_{s1} xor r_{s2}$	Y2
div	$r_d = r_{s1}/r_{s2}$	Y2
mul	$r_d = r_{s1} x r_{s2}$	Y2
shl	$r_d = r_{s1} << r_{s2}$	Y2
shr	$r_d = r_{s1} >> r_{s2}$	Y2
shra	$r_d = r_{s1} >> r_{s2}$	Y2
sub	$r_d = r_{s1} - r_{s2}$	Y2
min	$r_d = min(r_{s1}, r_{s2})$	Y2
max	$r_d = max(r_{s1}, r_{s2})$	Y2
chs	$r_d = -r_{s1}$	Y1
not	$r_d = \ r_{s1}$	Y1
abs	$r_d = \|r_{s1}\|$	Y1
com	$r_d = max(r_{s1}, r_{s2})$	Y2
mod	$r_d = max(r_{s1}, r_{s2})$	Y2
brv	Verilen yazmaçtaki bitleri ters sırada hedef yazmaca yazar	Y1
bfr	Verilen yazmacın belirtilen kadar kısmını maskeleyip hedef yazmaca yazar	Y1
br	Karşılaştırma bayraklarında belirtilen koşul varsa, verilen adres kadar ileri atlar	Dallanma
fin	Programı sonlandırır	Sistem
ldshr	Paylaşımlı bellekten yükleme işlemi yapar	Y1
stshr	Paylaşımlı belleğe saklama işlemi yapar	Y1
sync	Tüm threadler aynı noktaya gelinceye kadar önce gelen threadleri bekletir.	Sistem

Sonraki sayfada devam etmektedir.

Tablo 4.1 – devam

Buyruk	Açıklama	Türü
ldram	Ana bellekten yükleme işlemi yapar	Y1
stram	Ana belleğe saklama işlemi yapar	Y1
mov	$r_d = r_{s1}$	Taşıma
jmp	Program sayacına belirtilen sayıyı ekleyerek atlar	Atlama

Tosun buyruk kümesinde toplam 56 adet buyruk belirlenmiştir. Tablo 4.1 içinde verilen buyruklar içerdikleri işlenen tür ve sayılarına göre türlere ayrılmıştır. Bu sınıflandırma buyruk içinde belirtilmesi gereken işlenen cins ve sayılarına göre yapılmıştır. Buyruk türlerinin bit yapısının belirlenebilmesi için öncelikle buyruk içine yerleştirilecek bilgilerin bit genişlikleri belirlenmelidir.

Buyruk bit yapılarında kaynak ve hedef hafıza birimleri olarak yazmaç numaraları kullanılır. Buyruk içinde bir yazmacın kaç bit ile ifade edileceği, bir thread için tahsis edilen yazmaç sayısına bağlıdır. İşlemci mimarisinde yazmaç sayısının belirlenmesi bir ödünleşimli karardır. Yazmaç sayısının artması yazmaçlar için kullanılan alanı artıracağı gibi yazmaç numaraları için kullanılan karşılaştırıcı devrelerin de büyümesine sebep olur. Öte yandan yazmaç sayısının azlığı bellek işlemlerinin artmasına ve başarımın düşmesine sebep olacaktır. Tosun mimarisinde çok çekirdekli bir mimariden söz edildiği için yazmaç sayılarının artışı tek çekirdekli işlemcilere oranla daha fazla bir alan kullanımında artışa sebep olmaktadır. Bu yüzden Tosun mimarisinde hedef programlara yetebilecek minimum sayıda yazmaç kullanılmıştır. Bu çalışmada NVidia CUDA ile çalışan 184 adet paralel hesaplama uygulamasının yazmaç kullanım adetleri incelenmiştir. Elde edilen sonuçlara göre Tablo 4.2'ta sunulduğu şekilde 64 adetten fazla sayıda yazmaç kullanan program ile karşılaşılmamıştır.

Tablo 4.2: NVidia GPGPU Programları Yazmaç Kullanım Analizi

Açıklama	Adet
32 veya daha az sayıda yazmaç kullanan uygulamalar	138
32 ile 64 adet arasında yazmaç kullanan uygulamalar	46
64 yazmaçtan fazla sayıda yazmaç kullanan uygulamalar	0

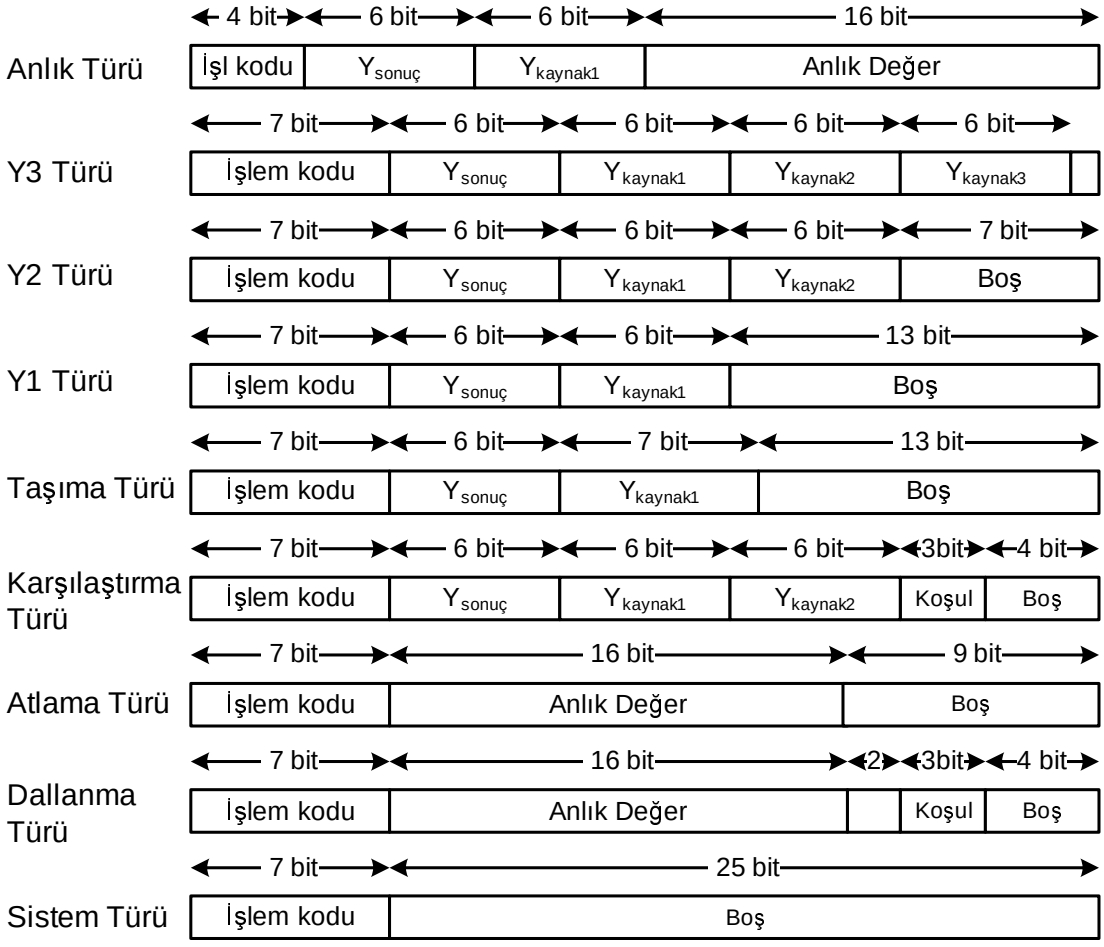
Neticede her bir thread için 64 adet yazmaçtan oluşan yazmaç öbeği kullanılmasına karar verilmiştir. Projenin bir diğer isteği olan OpenCL desteği ise OpenCL spesifikasyonlarında belirtilen bazı özel amaçlı yazmaçların gerçeklenmesini zorunlu kılmaktadır. Lokal thread numarası ve global thread numarası gibi programcının erişimine açık olması gereken ve spesifikasyonda belirtilen bilgiler program içinde özellikle adres hesaplamalarında sıklıkla kullanılmakta olduğundan yazmaç öbeğinde tutulması faydalı olacaktır. Bu bilgilerin yanı sıra program parametrelerinin de yazmaç öbeğine dahil edilmesi ile yazmaç sayısı 128 adete çıkarılmıştır. Ancak 128 yazmacın yalnızca ilk 64 adedi genel amaçlı olup, son 64 adeti özel mov buyruğu ile erişilebilir olarak belirlenmiştir. Toplamda 64 adet genel amaçlı yazmaç, buyruk içinde 6 bit ile ifade edilebilir.

Tüm işlemler 32 bit genişliğinde float veya tam sayılar ile yapılmaktadır. Yazmaç sayıları ve işlem kodu da hesaba katıldığında genel olarak buyrukların 32 bit genişliğe sığdırılabileceği hesaplanmıştır. Buyruklar için ayrılan bellek alanının verimli kullanılabilmesi için buyruk genişliklerinin de 32 bitten fazla olmamasına karar verilmiş, bu sebeple de buyruk içinde verilen anlık değerler 16 bit genişliğine sabitlenmiştir. Bir yazmaca anlık bir değerin yazılması ise movi ve movhi buyruklarının peş peşe kullanılması ile mümkündür.

Anlık türü buyruklar bir kaynak yazmacı, bir hedef yazmacı ve bir anlık değer içerir. Dolayısıyla işlem kodu için yalnızca 4 bitlik boş yer kalır. 4 bit, işlem koduna yeterli olmadığı için, olası tasarım çözümleri anlık değerin daraltılması veya buyruk genişliğinin artırılmasıdır. Buyruk genişliğinin değiştirilmesi durumunda bellek yönetimi, buyruk çekme ve kod çözme donanımları karmaşıklaşırken anlık değerin daraltılması durumunda ilave buyruklar gerekeceği gibi, programcının da tasarımı karmaşıklaşmaktadır. Bu probleme özel bir çözüm olarak anlık türü buyrukların 4 bit işlem koduna sahip olmasına karar verilmiştir. Anlık buyruklarda işlem koduna 4 bit ayrılmış olması, işlem kodunun kalan alt bitlerinin x ile doldurulması anlamına gelir. Toplamda 9 adet anlık türü buyruk bulunmaktadır. Dolayısıyla üst 4 biti [0,9] aralığında olan işlem kodları anlık türünde, [10,15] aralığında olan işlem kodları ise diğer türlerdedir. Buyruk kümesinde anlık türü olmayan, 46 adet buyruk vardır. Üst 4 bit için kullanılmayan 6 farklı değer olduğundan alt bitler için 8 farklı değer, dolayısıyla 3 bit gereklidir.

Anlık türü buyruklardan kaynaklı bu değişiklik ile Tosun buyrukları 7 bit işlem

kodu ile ifade edilir, 0000000 - 1001111 aralığındaki işlem kodları anlık türü buyruklara karşılık gelir, anlık türü buyruklarda alt 3 bit önemsiz olarak kabul edildiğinden yalnızca üst 4 bit buyruk içinde yer alır. Örneğin 0000xxx işlem kodu addi buyruğuna karşılık gelir. Dolayısıyla alt 3 bit buyruğun bit dizisi içinde yer almaz ve gelen herhangi bir buyruk için üst 4 bit 0000 ise buyruğun addi olduğu anlaşılır. Tüm buyruk türlerinin bit yapısı Şekil 4.1'ta sunulmuştur.



Şekil 4.1: Tosun Buyruk Türleri

4.2 Hesaplama Modülleri

Buyruk listesinde her bir buyruk için mimariye eklenmesi gereken hesaplama modülleri irdelenmiş, her bir buyruk için verimin yüksek tutulması adına ilgili optimize edilmiş Xilinx IPCore kullanımına öncelik verilmiştir.

- add, addi, sub, subi, abs, chs buyruklarının hesaplamaları tamsayı toplayıcı ipcore kullanılarak yapılır. Bu işlem birimi hem toplama hem çıkarma işlemini gerçeklemektedir.
- mul ve muli buyrukları integer çarpma IPCore kullanılarak gerçeklenir.
- and, andi, or, ori, not, xor, xori, brv ve bfr buyrukları mantıksal bit işlemleri yaparlar. Bu buyrukların her biri için ayrı bir işlem modülü kullanılır.
- min, max ve com buyrukları için iki sayının karşılaştırılması gerekmektedir. Bu üç buyruğun bir karşılaştırıcı modülünü kullanır. Com buyruğu işlem neticesinde büyük, küçük ve eşit bayraklarının değerini değiştirirken min ve max işlemleri sayılardan küçük olanı veya büyük olanı sonuç yazmacına yazar.
- div, divi ve mod buyrukları bölme işlemi için hazırlanmış ipcore kullanırlar.
- shl, shr, shra buyrukları kaydırıcı modül kullanılarak gerçeklenirler.
- f2int ve int2f buyrukları float ve integer veri tipleri arasında dönüşüm sağlar. Her ikisi için de hazır IPCore gerçeklenir.
- fadd ve fsub buyrukları için float toplayıcı IPCore kullanılarak gerçeklenir.
- fabs ve fchs buyrukları IEEE754 standardında işaret bitinin değiştirilmesi ile sağlanabilir. Bu iki buyruk için tek bir bit operasyon modülü gerçeklenir.
- fcom, fmin ve fmax işlemleri floating point bir karşılaştırıcı IPCore kullanırlar.
- fdiv ve fmod işlemleri floating point bir bölücü IPCore kullanılarak gerçeklenir.
- fexp e^x hesabı yapan IPCore kullanılarak gerçeklenir.
- ffma ve ffms işlemleri floating point fused multiply add IPCore kullanılarak gerçeklenir.
- fln buyruğu floating point doğal logaritma IPCore kullanılarak gerçeklenir.
- fmul buyruğu floating point çarpma IPCore kullanılarak gerçeklenir.

- fsqrt buyruğu floating point karekök IPCore kullanılarak gerçeklenir.
- fsin ve fcos buyrukları trigonometri IPCore kullanılarak gerçeklenir.

4.3 Boru Hattı Mimarisi

Buyruk kümesinde bulunan her buyruğun çalıştırılması sırasında geçmesi gereken sabit adımlar vardır. Öncelikle bir buyruk bellekten çekildikten sonra işlem kodu okunmalı ve uygun şekilde bitler ayrılarak buyruk içinde gelen yazmaç numaraları, anlık değerler vb. ayrıştırılmalıdır. Sonrasında ilgili yazmaçlarda tutulan değerler okunmalı, buyruk ile ilgili işlem seçilip okunan değerler üzerine uygulanmalı ve son olarak sonuç yazmacına sonuç yazılmalıdır. Bu adımlar arasına flip floplar eklenerek bir buyruğun adımları ardışık saat vuruşlarında takip etmesi sağlanabilir. Böylece bir buyruğun geçtiği adımdaki donanımlar boşa çıkar ve söz konusu buyruk tüm işlemleri tamamlamadan yeni bir buyruk aynı donanımları kullanarak hesaplamaya girebilir. Boru hattı tasarımında kaynakların etkin kullanımı son derece önemlidir. Eğer programın genelinde tüm boru hattı aşamaları aynı anda doldurulamıyorsa boru hattı kullanmanın avantajı yoktur. Öte yandan boru hattı aşamaları etkin bir şekilde doldurulabilirse buyruklar birbirinin çalışma sürelerini gizlerler ve her saat vuruşunda yeni bir sonuç üretilmiş olur.

Boru hattı aşamalarının tam doldurulması konusunda güncel problemlerin başında veri bağımlılıkları gelir. Eğer n. buyruğun kullanacağı bir veri m. buyruk tarafından hesaplanıyorsa, m. buyruk sonucu yazmaç öbeğine yazmadan n. buyruk yazmaç değerlerini okuyamaz. Veri bağımlılığı önlenemeyen bir problemdir. Bunun yerine literatürde veri bağımlılığı olmayan buyrukların, bekleyen buyrukların önüne alınması yöntemiyle çözülmektedir. Bu yaklaşıma "Out of order execution" ismi verilir. [20] [21]

Sırasız çalıştırma yöntemi beraberinde yazmaçların analizi, veri bağımlılıklarının çözülmesi, yazmaçların donanım seviyesinde yeniden adlandırılması, yazmaç sayıları ile ilgili bir sanallaştırma katmanı tanımlanması gibi donanımsal karmaşıklıkları da beraberinde getirmektedir. Oysa ki aynı anda çok fazla threadin koşturulacağı bir işlemcide, boru hattının etkin kullanımı için daha sade bir çözüm

olarak aralıklı işlem modeli kendini gösterir. [22]

Aralıklı İşlem Modeline göre çalışan işlemciler her bir buyruğun çalıştırılmasında sonra farklı bir thread'e geçiş yaparak çalışırlar. Çok sayıda birbirinden bağımsız işlemi bir arada yürütmeye çalışan işlemciler için Aralıklı İşlem tercih edilen bir yöntemdir [23] [24]. Bu şekilde çalışan işlemciler her bir thread için ayrı yazmaç öbeği ve program sayacı tutar. Herhangi bir thread'den boruhattına buyruk ataması yapıldığı zaman, farklı bir thread seçilerek bir sonraki buyruk o thread'in program sayacının gösterdiği yerden çekilir.

Aralıklı İşlem Modelinde veri bağımlılığı oluşmadığı için boruhattının etkin kullanımı sağlanmış olur. Farklı thread'ler arasında, yazmaç bazında, veri paylaşımı olmadığı için farklı thread'lerden buyrukların boruhattına alınması veri bağımlılığı sorunlarına yol açmaz. Böylece çok sayıda çevrim gerektiren buyruklar, farklı thread'lerden gelen buyrukların çalıştırılmasıyla gizlenmiş olur. Örnek vermek gerekirse, Tosun mimarisinde sin/cos işlemleri 28 saat vuruşunda tamamlanmaktadır. Tek bir thread üzerinden çalışan bir sistem düşünülürse bu sin/cos buyruğundan sonra gelen ve bunun sonucunu kullanan buyruk sin/cos'un tamamlanmasını beklemek zorunda kalır. Bu uzun süre içerisinde de boru hattının büyük bir bölümü boşta bekler. Aralıklı İşlem Modelinde ise aralarında veri bağımlılığı olma ihtimali olmadığı için farklı thread'lerden gelen buyruklar boruhattının içine alınabilir. Böylece sin/cos veya diğer çok sayıda saat vuruşunda sonuç veren işlemler için geçen süre başka buyrukların çalıştırılmasıyla gizlenmiş olur.

Aralıklı işlem modelinin bir sonucu olarak farklı threadler arasında hızlı bir şekilde "context switch" yapmak gerekmektedir. Yani bir thread çalışırken bir anda farklı bir threade geçilebilmesi gerekmektedir. Klasik işlemcilerde tüm yazmaç verilerinin belleğe kaydedilmesi ve diğer threade ait verilerin bellekten kopyalanmasıanlamına gelen context switch oldukça pahalı bir işlemdir. Oysa ki aralıklı işlem modelinden faydalanabilmek için 1 saat çevriminde context switch yapılması gerekmektedir. Bu hızda bir context switch ancak farklı threadlere ait yazmaçların da yazmaç öbeğinin bir kısmında saklanması ile mümkün olur. Tosun mimarisinde bu işlemin nasıl yapıldığı "Yazmaç Öbeği" başlığı altında anlatılacaktır.

Aralıklı işlem modeli ile çalışan Tosun boru hattı mimarisinin aşamaları şekil 4.3'de gösterilmiştir.

width=50

Şekil 4.2: Tosun Boru Hattı Mimarisi

4.3.1 Warp Seçimi

Warp NVidia tarafından literatüre kazandırılmış bir terimdir. Threadlerin bir araya toplanması ile oluşan thread grubuna warp ismi verilmiştir. Thread sözlükte ipliğe karşılık gelirken warp da dokumacılıkta kullanılan çözgü anlamını taşımaktadır. N adet threade sahip bir uygulamanın M adet SIMD Lane kapasitesi bulunan bir işlemcide çalıştırılması senaryosunda 3 farklı ihtimal vardır. N = M ise her bir SIMD lane üzerinde bir thread koşturulur. N < M ise bazı SIMD lane'ler boş kalır ve bunların sonuçları değerlendirilmez. En sık rastlanan durum olan N > M olması durumunda ise N adet thread M adet kapasiteli alt gruplara bölünür ve bir seferde M adet thread çalıştırılır. Arkasından ikinci ve üçüncü M adet thread barındıran gruplar çalıştırılır. Burada her M adet thread'den oluşan gruba warp ismi verilir. Dolayısıyla warp kapasitesi donanımda tanımlı SIMD lane sayısına bağlı iken warp sayısı uygulamadaki toplam thread sayısının warp büyüklüğüne bölümü ile hesaplanır. Threadlerin warplara ayrılma işlemi derleyici tarafından yapılır.

Aralıklı işlem modelinin bir uygulaması olarak, bir SIMD lane'e her saat vuruşunda farklı bir warp'a ait bir thread atanır. Hangi warp'un seçileceği boru hattının "Warp Seçimi" aşamasında belirlenir. Bu seçim Round-Robin politikasına göre gerçekleştirilir. Her warp için durum bitleri tutulur. Bu bitler warp'un "yürütme için uygun", "çalışıyor", "tamamlandı" gibi durumlarını gösterir. Uygun olan warp'lardan biri seçilir ve bu warp'un numarası boru hattının bir sonraki aşamasına aktarılır. Seçilen warp, boru hattını tamamlamadan bir daha seçilememesi için durum bitleri değiştirilerek işaretlenir. Aynı warp'un bir kez daha boru hattına alınması thread'lerin bir sonraki buyruklarının işlenmesi anlamına gelir. Bir warp boru hattını tamamlamadan ikinci kez boru hattına alınmadığında ikinci buyruk da boru hattına girmemiş olacağından herhangi bir veri bağımlılığı kontrolüne gerek kalmaz.

4.3.2 Buyruk Çekme

Buyruk çekme aşamasında bir önceki aşamadan gelen warp id'nin sıradaki buyruğu bellekten çekilir. Program buyrukları harici RAM'de tutulur. Buyruklara erişim program akışı sebebiyle genel olarak sıralı ve aralıklı işlem modeline göre tekrarlı olduğu için RAM'den gelen buyrukları bir süre Buyruk Önbelleği yapısında tutmak bu aşamayı oldukça hızlandıran bir optimizasyondur. Buyruğun çekilmesi ile bu aşama tamamlanır ve buyruk bir sonraki aşamaya geçirilir.

4.3.3 Buyruk Çözme

Bu aşamada buyruk çözümlenerek hangi işlem biriminin kullanılacağı, hangi yazmaçların okunup, hangilerine yazılacağı belirlenir. Tüm buyrukların 32 bit olması, işlem kodu genişliklerinin buyruklar arasında fazla farklılık göstermemesi ve neredeyse tüm buyrukların aynı yazmaçlara erişim yapabilmesinden dolayı, boru hattının bu aşaması sade bir yapıdadır.

4.3.4 Yazmaç Çekme

Burada çalıştırılmak üzere olan buyruğun işlem sırasında kullanacağı verilen yazmaç öbeğinden alınır. Her bir SIMD lane üzerinde her bir warp için ayrı bir Yazmaç Öbeği vardır ve bunlardan kullanılacak veriler aynı anda çekilir. İki adet kaynak yazmacı bulunan buyruklarda ve 16 çekirdekli bir adada toplam 32(16*2) adet 32-bitlik veri ortalama 1 çevrimde okunur.

4.3.5 Hesap Modülü Atama

Boru hattının bu aşaması hesaplamanın başlatıldığı yerdir. Bu aşamaya gelen bir buyruğun tüm verileri hesaplamaya hazır bir halde beklemektedir. Bu aşamada işlem koduna bakılarak buyruk gerekli hesaplama donanımına gönderilir.

4.3.6 Hesap

Hesaplamanın yapıldığı aşamadır. Burada birçok işlem birimi yer alır. Bunlardan, sık kullanılan ve daha az alan kaplayan işlem birimleri SIMD lane adetindedir. Bu şekilde, bu işlem birimleri gelen tüm verileri aynı anda işleme sokabilecek durumdadır. Daha nadir erişilen trigonometrik işlemler ve logaritma gibi hesaplardan sorumlu işlem birimleri ise daha az sayıda bulunabilir. Az sayıda bulunan işlem birimlerinin kendi boru hattı mevcuttur. Örneğin SIMD lane sayısının yarısı adetinde olan bir hesaplama modülü ilk çevrimde gelen sayıların yarısını işleme alır, ikinci çevrimde ise diğer yarısını işleme alır. Böylece tüm sayılar boru hattında peşi sıra ilerlemiş olurlar. Örneğin 28 çevrim süren bir sinus işlemi için SIMD lane sayısının çeyreği kadar sinus hesaplama birimi yerleştirilmişse, tüm sayıların sinus sonuçlarının hesaplanması $28 + 3 = 31$ çevrim sürer. Alan kullanımı ve performans optimizasyonu için esneklik sağlayan bu yapıda ilave 3 çevrim kabul edilerek alandan kazanılabilir ya da hesap modülü sayısı artırılarak performans artışı sağlanabilir. Hesap aşamasının sonunda bir sonuç buffer'ı bulunmaktadır. Hesap modüllerinin boru hattından çıkan sonuçlar önce bu buffer'lara yazılır ve yazılmak için kendi sıralarının gelmesini beklerler.

4.3.7 Geri Yazma

Geri yazma aşaması sonuçların yazmaç öbeklerine yazıldığı aşamadır. Geri yazma aşamasının kontrolcüsü sürekli olarak hesap modüllerinin çıkışlarındaki sonuç buffer'larını kontrol eder ve sırasıyla sonuçları ilgili yazmaçlara yazar.

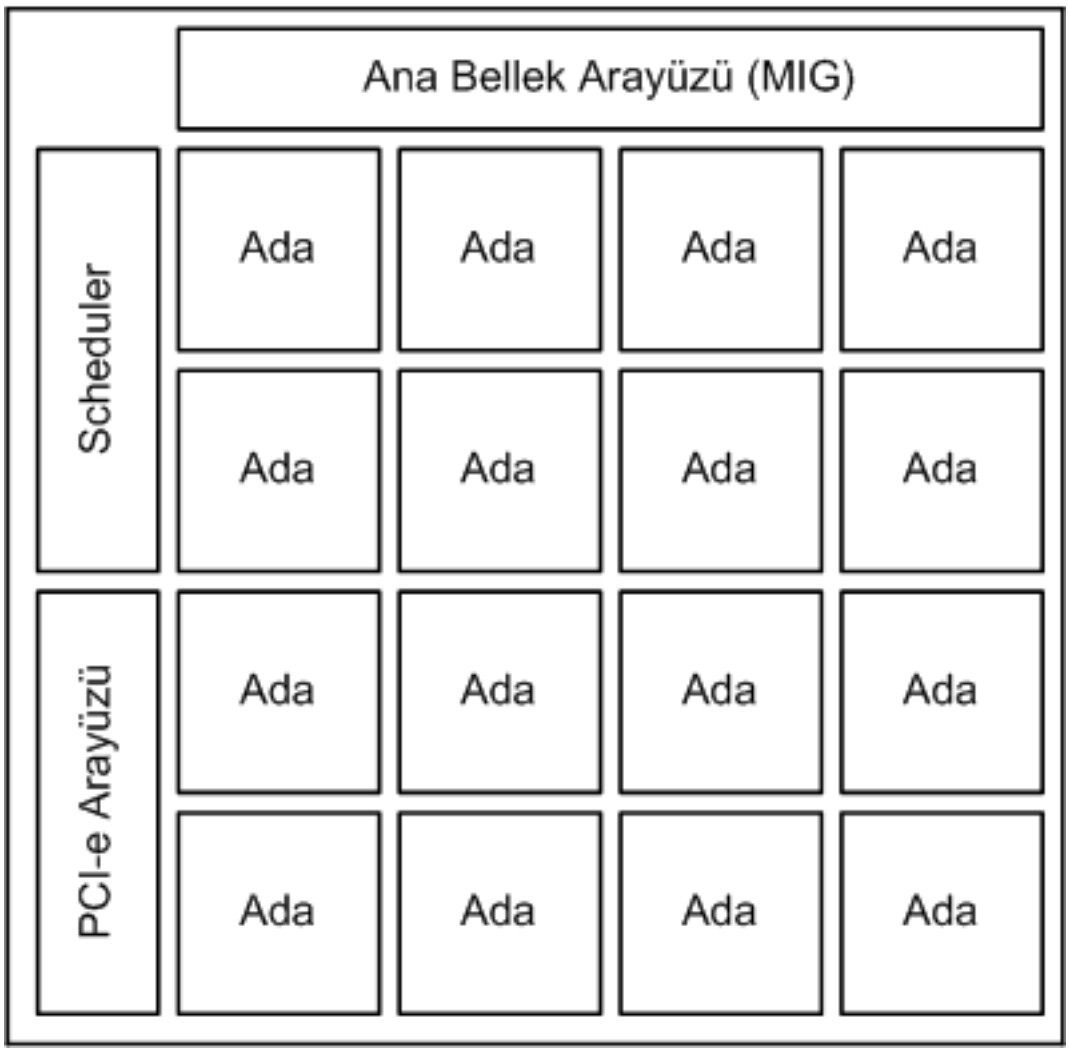
4.4 Veri Yolu Mimarisi

Önceki bölümlerde Tosun mimarisinin buyruk kümesi, hesaplama donanımları ve boru hattı aşamaları belirlenmiştir. Parçaların birleştirilmesi ile veri yolu mimarisi oluşacaktır. Tasarım gereksinimleri arasında belirtilen ölçkelenebilirlik özelliğinden dolayı tüm kodlama parametrik olarak yapılmıştır. Mimarinin üzerine inşa edildiği temel parametrelerden biri de SIMD lane sayısıdır. SIMD lane

sayısındaki artış, veri yolu genişliklerinin, karşılaştırıcı, kod çözücü ve kodlayıcı gibi donanımların katlanarak artmasına sebep olmaktadır. Bu etki hem alan kullanımında hem de sinyal gecikmelerinde artışa neden olur. Neticede performans kaygısı ile paralelliğin artması için SIMD lane sayısının artırılması ile alan kullanımı büyümekte, gecikmeler artmakta ve hem güç tüketimi artmakta hem saat sıklığı azalmaktadır. Dahası FPGA içi routing işlemi de SIMD lane sayısının artması ile zorlaşmakta ve imkansız hale gelebilmektedir. Bu etki, kaçınılmaz olmakla beraber hiyerarşik tasarım kullanılarak azaltılabilir.

Tosun mimarisi routing ve timing ile ilgili kısıtları zorlayabilmek adına hiyerarşik bir yapıda tasarlanmıştır. Doğrudan N adet SIMD lane gerçeklenmesi yerine küçük gruplar halinde, $N = N_1 x N_2$ olacak şekilde N_1 adet ada ve her adanın içinde N_2 adet SIMD lane olacak şekilde gerçeklenmiştir. Hiyerarşinin üst seviyesinde, ölçeklenebilirliği olmayan PCI-e ve Ana bellek arayüzü gerçeklenmiş ve AXI bus yapısı ile N adet ada ismi verilen donanıma bağlanmıştır. Tosun üst seviye mimari çizimi Şekil 4.4'da sunulmuştur. Mimarinin büyük tek bir ada yerine çok sayıda daha küçük adalardan oluşmasının iki sebebi vardır.

Her bir ada içindeki threadlerin veri paylaşabilmesi için ada içine yerleştirilen paylaşımlı belleğe erişimi olan çekirdek sayısı ile bu bellekte yaşanan gecikme doğrudan ilişkilidir. Paylaşımlı bellek açısından bakıldığında istemci sayısının artması istek paketlerinin beklediği kuyrukta uzamaya sebep olmaktadır. Bu durum sınav zamanında kütüphaneden kitap almak isteye öğrenciler analojisiyle açıklanabilir. Her bir öğrenci bir istek paketi, ulaşmak istedikleri kitaplar da paylaşımlı bellekte tutulan veriler olsun. Kütüphanede kitap ödünç alımıyla ilgilenen personel sayısını sabit olarak 2 kabul edelim. Kitap sayısı sonsuz bile olsa, N adet öğrencinin bu iki personel üzerinden kitaplara erişim imkanı varken, öğrenci sayısının artması ile bir öğrencinin ortalama kitaba ulaşma süresi doğru orantılı olarak artacaktır. Artışı engellemek için yapılabilecek iki seçenekten birincisi öğrenci sayısını sınırlamak, ikincisi ise personel sayısını artırmaktır. Bu analojide personel sayısı FPGA üzerinde gerçeklenen Block RAM'lerin port sayısını ifade eder. Block RAM'lerin port sayısı 2'den fazla olamadığından istemci sayısını azaltmak tek çözümdür. Bu bağlamda tasarımı adalara ayırmak, kütüphaneyi parçalamaya ve kütüphane başına düşen öğrenci sayısını azaltmaya benzer. Dolayısıyla çok sayıda çekirdeği küçük gruplar halinde ayırarak her gruba bir paylaşımlı bellek tahsis etmek bellek işlemlerinin performansını artıracaktır.



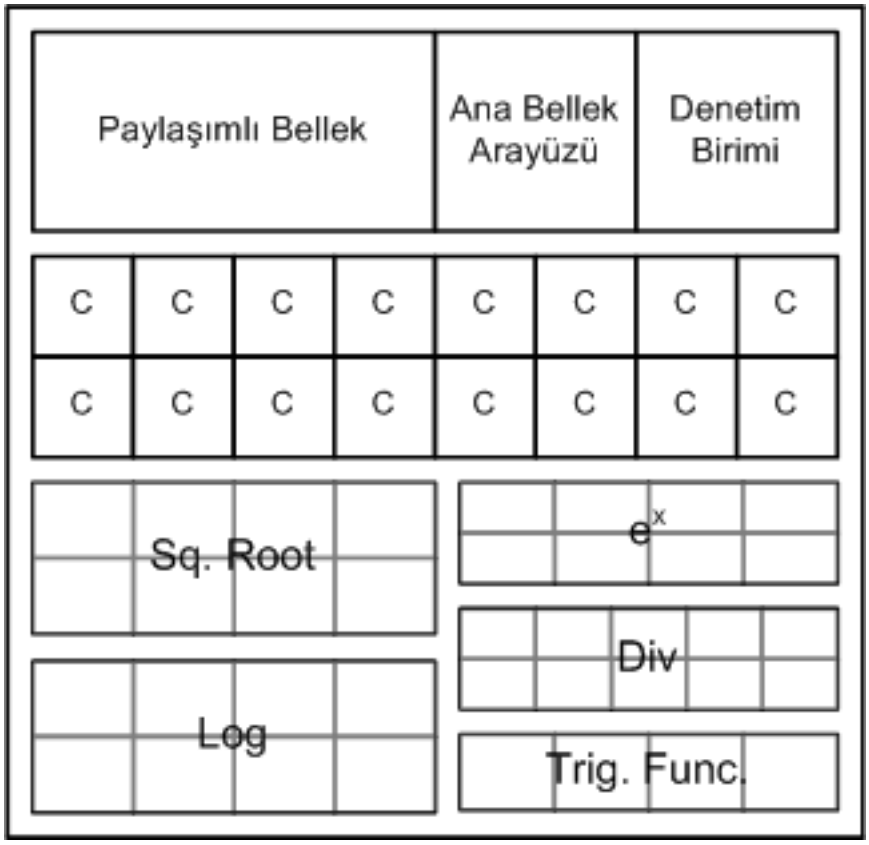
Şekil 4.3: Tosun Üst Seviye Mimarisi

Diğer bir sebep ise yukarıda bahsedilen, FPGA gerçeklemesi sırasında oluşabilecek timing ve routing problemleridir. Yapılan bir tasarım FPGA üzerinde gerçeklenirken herhangi bir kısıt tanımlanmamışsa birbirine yakın olması beklenen bazı donanım parçaları yonga üzerinde uzak yerlere denk gelebilir. Ölçeklenebilir tasarımlarda bu problem sıklıkla kaynakların verimsiz kullanımına ve tellerin uzaması ile kritik yollardaki gecikmelerin artmasına dolayısıyla saat frekansının düşmesine ve nihayetinde performans düşüşüne sebep olabilir. Bu problemlerden kaçınmak için sıklıkla hiyerarşik tasarımlar gündemde tutulur. Tosun tasarımının homojen adalardan oluşması sentez aracına hangi donanımların yakın olması gerektiği konusunda daha çok fikir vermekte ve bu konudaki potansiyel problemlerin indirgenmesine olanak sağlamaktadır.

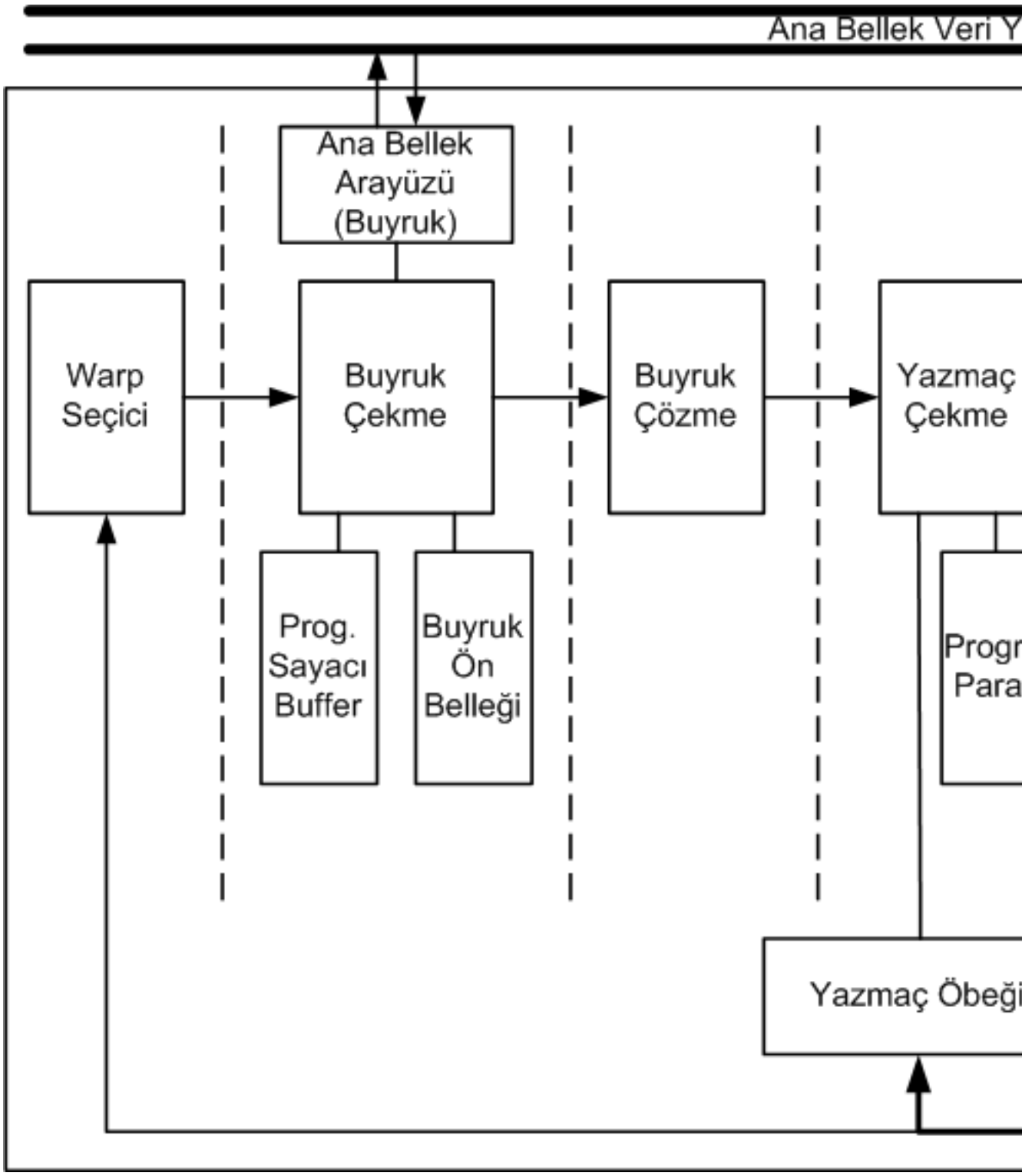
Tasarımın adalara ayrılması ile donanım seviyesinde bir soyutlama sağlanmıştır. Bu soyutlama, Tosun üzerinde o anda koşan tüm threadlerin aynı anda aynı buyruklarının koşması zorunluluğunu ortadan kaldırır. SIMD mimarinin bir özelliği olarak herhangi bir t anında ada içerisinde çalışan tüm threadlerin aynı buyrukları koşmakta, bütün threadler için o buyruk tamamlanmadan diğer buyruğa geçilmemektedir. Öte yandan aynı anda farklı adalarda farklı buyruklar çalışıyor olabilir. Bu sayede ana bellek erişimi farklı adalar için farklı zamanlarda gerçekleşebilir; böyle bir durumda beklemeler azalır. Farklı adaların farklı zamanlarda bellek erişimi istemesi ise ilk bellek erişiminden sonra kaçınılmazdır.

4.4.1 Ada Veri Yolu Mimarisi

Tüm işlemler için Özel İşlem Birimi yapısı aynıdır. Giriş ve çıkışlardan da sadece "sayılar" girişleri içerideki birime göre değişken olabilir, diğer tüm giriş ve çıkışlar ise standarttır. Örneğin; toplama birimi için 2 adet 32-bitlik giriş varken, multiply-add işlemi için 3 adet sayı gerekir. Şekil 28'de "N" hesaplamada kullanılan eleman sayısını göstermektedir. Özel İşlem Birimleri Şekil 28'de sağ tarafta gösterildiği gibi Özel İşlem Grubu'nu oluşturur. Burada en fazla çekirdek sayısı kadar olmak üzere değişken sayıda işlem birimi yer alabilir. Grup içerisindeki işlem birimlerinin paylaştığı "Result Buffer" birimi vardır. Result Buffer yine Özel İşlem Grubu içerisinde yer alan "sayaç" ile birlikte bir FIFO yapısı gibi davranır. Özel İşlem Birimlerinden çıkan sonuçlar Result Buffer'a yazılır ve "Write-Back" biriminin bu verileri okuyup yazmaç öbeğine yazması beklenir. Write-Back birimi her bir Özel İşlem Grubuna bir öncelik atayarak hazır olan sonuçları bu önceliklere göre yazmaç öbeğine geri yazar.



Şekil 4.4: Tosun Ada Mimarisi (Kavramsal)



Şekil 4.5: Tosun Ada Mimarisi (Boru Hattı)

5. ALT MODÜLLERİN TASARIMI

6. SONUÇ

Kaynakça

[1] Edwin. J. Tan, Wendi. B. Heinzelman. 2003. DSP Architectures: Past, Present and Futures. ACM Sigarch Computer Architecture News

[2] Hallmans, Daniel, et al. 2013. GPGPU for industrial control systems. IEEE 18th Conference on Emerging Technologies & Factory Automation ETFA

[3] Emmett Kilgariff and Randima Fernando. 2005. The GeForce 6 series GPU architecture. In ACM SIGGRAPH 2005 Courses SIGGRAPH '05, John Fujii (Ed.). ACM, New York, NY, USA

[4] Kirk, D. 2007. NVIDIA CUDA software and GPU parallel computing architecture. ISMM Vol. 7, pp. 103-104

[5] Stone, J. E., Gohara, D., & Shi, G. 2010. OpenCL: A parallel programming standard for heterogeneous computing systems. Computing in science & engineering, 12(3), 66

[6] Kuon, I., & Rose, J. 2007. Measuring the gap between FPGAs and ASICs. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 26(2), 203-215.

[7] Smith, R.L., T̈he MATLAB project book for linear algebra; 1997 Prentice Hall

[8] Gotze, J.; Paul, S.; Sauer, M., Än efficient Jacobi-like algorithm for parallel eigenvalue computation,C̈omputers, IEEE Transactions on , vol.42, no.9, pp.1058,1065, Sep 1993

[9] Flynn, M. J. (September 1972). S̈ome Computer Organizations and Their Effectiveness: IEEE Trans. Comput. C–21 (9): 948–960. doi:10.1109/TC.1972.5009071

[10] Shivakumar, P., Kistler, M., Keckler, S. W., Burger, D., & Alvisi, L. (2002). Modeling the effect of technology trends on the soft error rate of combinational logic. In Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on (pp. 389-398). IEEE.

[11] Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., ... & Hanrahan, P. (2008). Larrabee: a many-core x86 architecture for visual computing. ACM Transactions on Graphics (TOG), 27(3), 18.

[12] Molka, D., Hackenberg, D., Schone, R., & Muller, M. S. (2009, September). Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system. In Parallel Architectures and Compilation Techniques, 2009. PACT'09. 18th International Conference on (pp. 261-270). IEEE

[13] Hackenberg, D., Molka, D., & Nagel, W. E. (2009, December). Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems. InProceedings of the 42Nd Annual IEEE/ACM International Symposium on microarchitecture (pp. 413-422). ACM

[14] Heinecke, A., Klemm, M., Bungartz H.J., T̈rom GPGPU to Many-Core: Nvidia Fermi and Intel Many Integrated Core ArchitectureC̈omputing in Science and Engineering, vol. 14, no. 2, pp. 78-83, March-April, 2012

[15] http://supercomputingblog.com/cuda/cuda-memory-and-cache-architecture/

[16] Wentzlaff, D., Griffin, P., Hoffmann, H., Bao, L., Edwards, B., Ramey, C., Mattina, M., Miao, C.-C., III, J. F. B. & Agarwal, A. (2007). On-Chip Interconnection Architecture of the Tile Processor.. IEEE Micro, 27, 15-31.

[17] http://docs.nvidia.com/cuda/parallel-thread-execution/#texture-instructions

[18] http://en.wikipedia.org/wiki/X86_instruction_listings

[19] http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html

[20] Gieseke, B. A., Allmon, R. L., Bailey, D. W., Benschneider, B. J., Britton, S. M., Clouser, J. D., ... & Wilcox, K. E. (1997, February). A 600 MHz

superscalar RISC microprocessor with out-of-order execution. In Solid-State Circuits Conference, 1997. Digest of Technical Papers. 43rd ISSCC., 1997 IEEE International (pp. 176-177). IEEE.

[21] Garg, S., Hagiwara, Y., Lau, T. L., Lentz, D. J., Miyayama, Y., Trang, Q. H., ... & Wang, J. (1996). U.S. Patent No. 5,560,032. Washington, DC: U.S. Patent and Trademark Office.

[22] Laudon, J., Gupta, A., & Horowitz, M. (1994). Interleaving: A multithreading technique targeting multiprocessors and workstations. ACM SIGPLAN Notices, 29(11), 308-318.

[23] Control Data Corp, «CDC Cyber 170 Computer Systems; Models 720, 730, 750, and 760; Model 176 (Level B); CPU Instruction Set; PPU Instruction Set,» pp. 2-44.

[24] A. e. a. Snavely, M̈ulti-processor Performance on the Tera MTA,ïn IEEE Computer Society Proceedings of the 1998 ACM/IEEE conference on Supercomputing, 1998.

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, Adı	: Yağlıkçı, Abdullah Giray
Uyruğu	: T.C.
Doğum tarihi ve yeri	: 09.08.1988 Ankara
Medeni hali	: Bekar
Telefon	:
Faks	:
e-mail	: agyaglikci@.etu.edu.tr

Eğitim

Derece	Eğitim Birimi	Mezuniyet Tarihi
Y. Lisans	TOBB Ekonomi ve Teknoloji Üniversitesi	2014
Lisans	TOBB Ekonomi ve Teknoloji Üniversitsi	2011

İş Deneyimi

Yıl	Yer	Görev
2012-2014	TOBB Ekonomi ve Teknoloji Üniversitesi	Eğitim Asistanı
2010-2012	Yumruk Uzay ve Savunma Teknolojileri Ltd	Elektronik Tasarım Mühendisi

Yabancı Dil

İngilizce (Çok iyi)
Rusça (Çok kötü)
Arapça (Çok kötü)

Yayınlar