Gömülü Sistemler İçin Karşılaştırma Uygulaması Geliştirme

Etem Deniz ve Alper Şen
Boğaziçi Üniversitesi
Bilgisayar Mühendisliği Bölümü
Bebek, 34342, İstanbul
e-posta: {etem.deniz, alper.sen}@boun.edu.tr

***Özetçe*—Karşılaştırma uygulamaları, gerçek uygulamaların özelliklerini taşırlar ve yeni donanımlar geliştirirken performans ve enerji analizinde yaygın şekilde kullanılırlar. Sentetik karşılaştırma uygulamaları, karşılaştırma uygulamalarının yüksek hızda simülasyona olanak sağlayan minyatür halleridir. Bu çalışmada, gerçek uygulamalardan yazılım mimari kalıplarını kullanarak çok çekirdekli gömülü sistemler için sentetik karşılaştırma uygulamaları geliştirilmektedir. Sentetik karşılaştırma uygulamalarını otomatik şekilde yaratmak için karakterizasyon, kalıp tanımlama ve sentezleme bileşenleri olan bir araç geliştirdik. Eskiden geliştirilmiş sentetik karşılaştırma uygulamalarından farklı olarak bizim geliştirdiklerimiz herhangi bir altyapıda (simetrik çok çekirdekli sistemler, mesaj iletim sistemleri, vb.) koşabilir. Bu sayede sentetik karşılaştırma uygulamaları, homojen ve heterojen çok çekirdekli gömülü sistemler için uygundur. Geliştirdiğimiz aracı kullanarak PARSEC karşılaştırma uygulamaları üzerinde deneyler yaptık. Deney sonuçlarımız, geliştirdiğimiz sentetik karşılaştırma uygulamaları ile gerçek uygulamaların birçok mikro-mimari bağımlı ve bağımsız metrik açısından benzer olduğunu gösterdi.**

I. GİRİŞ

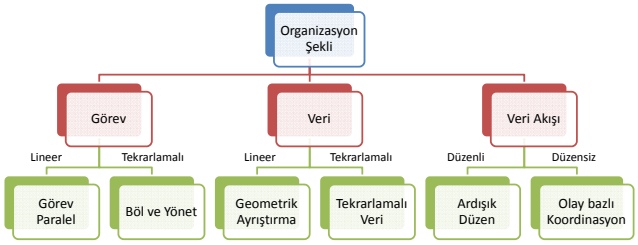
Endüstride çok çekirdekli işlemcilere eğilim artmaktadır çünkü tek çekirdekli işlemciler karmaşıklıkta ve hızda fiziksel limitlerine ulaşmaktadır. Gömülü çok çekirdekli sistemler sağlık, otomotiv ve bilgisayar ağları gibi birçok alanda kullanılmaktadır. Ancak koşutzamanlılığın doğasındaki karmaşık ve analizi çetin davranışlardan dolayı çok çekirdekli donanım geliştirmek zordur. Karşılaştırma uygulamalarının yokluğunda, çok çekirdekli sistemlerin geliştirilmesi daha da zor olur. Karşılaştırma uygulamaları birçok gerçek uygulamanın davranışlarını taşımaktadır ve yeni sistemlerin tasarımında ve performans analizlerinde kullanılırlar. Örneğin, PARSEC [1] karşılaştırma uygulamaları paketinde paylaşımlı bellek mimarileri için çok izlekli uygulamalar vardır, Rodinia [2] karşılaştırma uygulamalarında, heterojen sistemlerler için uygulamalar ve EEMBC [3] karşılaştırma uygulamalarında ise gömülü sistemler için uygulamalar vardır.

Karşılaştırma uygulamaları sayesinde yeni donanım mimarileri ve bunların performanları, tasarımın ilk aşamalarında tahmin edilebilir. Bu tahminleri yapmak için yüksek simülasyon hızlarına ihtiyaç vardır ancak mevcut müşteri uygulamaları ve karşılaştırma uygulamaları büyük ve yavaştır. Bu yüzden yüksek simülasyon hızlarına erişmek için hızlı çalışan karşılaştırma uygulamalarına ihtiyaç vardır. Bunun yanında, geleneksel karşılaştırma uygulama paketlerinden PARSEC ve Rodinia, paylaşımlı bellek mimarilerine, Pthreads, OpenMP, OpenCL kütüphanelerine ve tekdüze işlemci mimarilerine uygundur. Ancak çok çekirdekli heterojen gömülü sistemler, bu mimarileri ve kütüphaneleri desteklemedikleri için bu geleneksel karşılaştırma uygulamalarını kullanamayabilirler. Bu noktada, birçok altyapıya (simetrik çok çekirdekli işlemciler (SMP), mesaj iletim mimarileri, heterojen sistemler, vb.) uygun karşılaştırma uygulamalarının geliştirilmesine ihtiyaç vardır.

Yukarıdaki problemlere çözüm olarak bu çalışmada çok çekirdekli gömülü sistemler için herhangi bir altyapıya uygun sentetik karşılaştırma uygulaması geliştirilmektedir. Karşılaştırma uygulamaları, faydalı hesaplama yapmayan ancak gerçek uygulamaların karakteristikleri ya da özelliklerini taklit edebilen küçük, basit ve hızlı uygulamalardır. Multicore Association (MCA) [4], heterojen gömülü sistemlere uygun uygulama geliştirmek için temel bir yapı (kütüphane) sunmaktadır. Geliştirdiğimiz sentetik karşılaştırma uygulamaları taşınabilir bir katman sağlayan MCA API (Multicore Communications API (MCAPI) veya Multicore Resource API (MRAPI)) kullanmaktadır. Bu özellik sayesinde, geliştirdiğimiz sentetik karşılaştırma uygulamalarının gerçek uygulamalara benzerliklerini, SMP veya paylaşımlı bellek mimarisine ihtiyaç olmadan gösterebiliyoruz.

Sentetik karşılaştırma uygulaması geliştirme, verilen gerçek uygulamanın özelliklerinin toplanması ile başlar. Uygulama karakteristikleri iki guruba bölünebilir: mikro-mimari bağımsız (üst seviye) ve mikro-mimari bağımlı (alt seviye). Komut seviyesi paralellik (instruction level parallelism), veri yerelliği (data locality) üst seviyeye ve önbellek kaçış oranı (cache miss rate), hatalı dallanma öngörüsü (branch misprediction)) alt seviyeye örneklerdir.

Bu çalışmada, çok çekirdekli uygulamarın üst seviye karakteristiklerini koruyan paralel tasarım kalıplarını kullanılmaktadır. Yazılım mimarisi, yazılımın bileşenlerini, bileşenlerin rollerini ve aralarındaki etkileşimi tanımlar. Kalıplar, yazılım mimarilerini sistematik olarak tanımlar. Yazılım kalıpları, sıklıkla oluşan problemler için yüksek kalitelide çözüm sunar ve yeni uygulama geliştirmeyi kolaylaştırır. Çok izlekli (multi-threaded) paralel uygulamaları hedeflediğimiz için paralel tasarım kalıplarını mikro-



Şekil 1. Yazılım için paralel tasarım kalıpları [5]

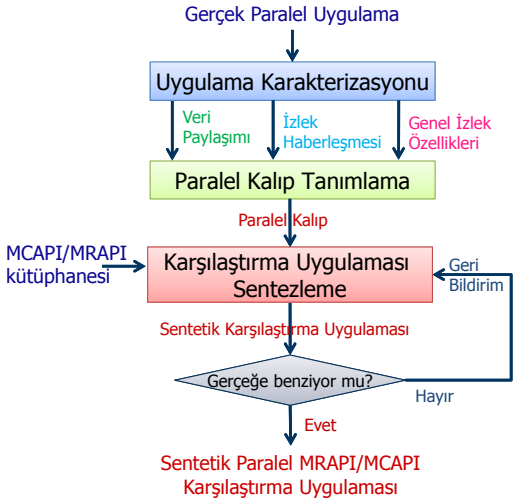
mimari bağımsız karakteristik olarak kullanıyoruz. Mikro-mimari bağımlı metriklerin birçogu daha önceden yapılan benzerlik karşılaştırmalarında kullanılmıştır fakat yazılım tasarım kalıplarını bizden başka kullanan olmamıştır. Paralel tasarım kalıpları (örneğin ardışık düzen (pipeline), böl ve yönet (divide and conquer)), Mattson ve diğerleri [5] tarafından tanımlanmıştır. Tanımlanan bu paralel tasarım kalıpları, karşılaştırma uygulaması geliştirmede üst seviye iskelet sağlamaktadır.

Heterojen çok çekirdekli gömülü sistemler için elle karşılaştırma uygulaması geliştirmek, hata yapmaya açık ve yoğun emek isteyen bir süreçtir. Verilen uygulamayı analiz etmek ve ondan sentetik karşılaştırma uygulaması yaratmak için otomatik bir araç geliştirdik. Paralel tasarım kalıplarının sentetik karşılaştırma uygulaması geliştirmeye uygun olduğunu doğruladık. Geliştirdiğimiz sentetik karşılaştırma uygulamalarının, geliştirildikleri gerçek uygulamalara çeşitli mikro-mimari bağımlı ve bağımsız karakteristikler açısından benzer olduğunu göstermek için PARSEC karşılaştırma uygulamaları üzerinde deneyler yaptık. Geliştirdiğimiz karşılaştırma uygulamalarının ortalama %86 ve en yüksek %95 benzerlik taşıdıklarını gözlemledik. Ek olarak, sentetik uygulamalarımız C programlama dilinde oldukları için önceden alt seviye dillerde geliştirilenlere göre okunulabilirlikleri daha yüksektir.

II. YAZILIM MİMARİ KALIPLARI

Yazılım mimari kalıpları, yazılım sistemlerinde yaygın görünen üst düzey yapıların tanımlarıdır [6]. Yazılım tasarımı sırasında verilen en önemli kararlardan birisi yazılım mimari kalıbının seçilmesidir. Yazılım geliştiricileri, mimari kalıplar sayesinde karmaşık sistemlerin yapı bloklarını ve aralarındaki ilişkiyi anlayabilirler. Nesneye dayalı programlama için geliştirilen yazılım mimari kalıplarının çok kullanışlı oldukları görülmüştür [7]. Benzer şekilde, içerisinde paralel tasarım kalıplarını barındıran bir paralel mimari kalıplar dili geliştirilmiştir [8].

Şekil 1'de, yazılım tasarım kalıpları, karar ağacı üzerinde gösterilmektedir. Paralel tasarım kalıpları, görevlerin, verinin ve veri akış organizasyonuna göre üç ana sınıfa ayrılır ve her sınıfta ikişer tasarım kalıbı bulunur. *Görev paralel (task parallel)* kalıbında, problem birbirinden bağımsız görevler arasında paylaştırılır ve çözülür. *Böl ve yönet (divide and conquer)* kalıbında ise problem küçük alt-problemlere bölünür ve her alt-problem



Şekil 2. Sentetik karşılaştırma uygulaması geliştirme aracı

bağımsız olarak çözülür ve sonuçlar birleştirilir. Görev organizasyonlu kalıplarda, problem yerel veriler üzerinde çözülürken, veri organizasyonlu kalıplarda, global veri üzerinde çözülür. Problem çözümünde kullanılan global veri, görevler tasafından eşit olarak paylaştırılırsa, *geometrik ayrıştırma (geometric decomposition)* kalıbı, diğer durumda ise *tekrarlamalı veri (recursive data)* kalıbı olur. Matris çarpımı ve vektör işlemeri geometrik ayrıştırma kalıbının ve grafik arama ve ağaç algoritmaları tekrarlamalı veri kalıbının kullanımına örnektir. *Ardışık düzen (pipeline)* kalıbında, birbirinden bağımsız bir grup sıralı safhada problem çözülür ve her bir safhanın çıktısı bir sonraki safhanın girdisi olur. *Olay bazlı koordinasyon (event-based coordination)* kalıbında ise her bir olay yeni görevler tetikler ve görevlerin koşutzamanlı olarak çalışmasıyla problem çözülür.

III. KARŞILAŞTIRMA UYGULAMASI GELİŞTİRME ARACINA GENEL BAKIŞ

Şekil 2 geliştirdiğimiz otomatik sentetik uygulama geliştirme aracını üst seviyeden göstermektedir. Aracımızda üç ana birim vardır: *uygulama karakterizasyonu*, *paralel kalıp tanımlama* ve *karşılaştırma uygulaması sentezleme*. Uygulama karakterizasyonu birimi, dinamik ikili enstrümantasyon (dynamic binary instrumentation) aracı kullanarak uygulamanın karakteristiklerini toplar. Bu enstrümantasyon aracı sayesinde, çalıştırılabilir koda yeni kodlar ekleyerek ilgili karakteristikler kaydedilir. Paralel kalıp tanımlama birimi, toplanan karakteristiklerden uygulamanın tasarım kalıbını tanımlar. Tasarım kalıbı tanımlamak için literatürde yaygın olarak kullanılan k-en yakın komşu (k-nearest neighbor) tekniği kullanılır. Son olarak, karşılaştırma uygulaması sentezleme birimi, tasarım kalıbından ve toplanan karakteristiklerden bir sentetik karşılaştırma uygulaması yaratır. Daha sonra yaratılan sentetik ile gerçek uygulama arasındaki benzerlik hesaplanır. Eğer hedeflenen benzerlik oranına ulaşılmamışsa, benzer olmayan karakteristikler tespit edilerek yeni sentezleme

yapılır. Sentetiklerin heterojen gömülü sistemlere uygun olması için sentezleme sırasında tercih edilen MCA API (MCAPI veya MCAPI) kullanılır. Sonraki bölümlerde, her bir birim detaylı olarak anlatılmaktadır.

IV. PARALEL UYGULAMALARIN KARAKTERİZASYONU

Uygulama karakterizasyonundaki amaç uygulamanın nicel özelliklerinin toplanmasıdır. Çalışmamızda, paralel kalıp tanımlamaya olanak sağlayan üst seviye karakteristikleri kullanıyoruz. Uygulama karakteristiklerini üç grupta inceliyoruz: *veri paylaşımı (data sharing)*, *izlek haberleşmesi (thread communication)* ve *genel izlek özellikleri (general threading)*. Her grup alt-karakteristiklere sahiptir. Veri paylaşımındaki alt-karakteristikler *salt okunur (read-only)*, *gezici (migratory)*, *üretici/tüketici (producer/consumer)* ve *özel (private)* şeklindedir. Salt okunurda, veriler güncellenmez ve sadece okunur. Birden fazla izlek tarafından okunup yazılan veriler paylaşımlıdır. Paylaşımlı veri, bir izlek tarafından kısa bir süre içerisinde okunup yazılıyorsa ve bu durum birçok izlek tarafından tekrarlanıyorsa gezici veri; diğer durumda ise üretici/tüketici veri olur. Tek bir izlek tarafından okunup yazılan veriler özeldir. İzlek haberleşmesinin *hiç*, *biraz* ve *çok* olmak üzere üç alt-karakteristiği vardır. Haberleşen izleklerin sayısına ve haberleşmede kullanılan verinin miktarına göre izlek haberleşme karakteristiğine karar verilir. Son olarak, genel izlek özelliklerinde, *izlek sayısı*, *izleklerin koşma süreleri*, *program sayacı* ve *dinamik komut sayısı* alt-karakteristikleri vardır. Program sayacı izleklerin aynı fonksiyonu koşturup koşturmadığına ve dinamik komut sayısı izleklernin iş yükünün dengeli dağılıp dağılmadığına karar vermede kullanılır. Ek olarak, *izleklerin yaratıcıları*nı, *koşmaya başlama ve çıkış zamanları*nı gösteren bir ilişki grafiği oluşturuyoruz.

V. PARALEL KALIPLARIN TANIMLANMASI

Karakterizasyon aşaması tamamlandıktan sonra, toplanan karakteristikler kullanılarak k-en yakın komşu tekniğiyle uygulamanın paralel kalıbı tanımlanır. Bu teknikte, modelin oluşturulması ve kullanılması olmak üzere iki adım vardır. İlk adımda, modeli oluşturmak için her bir kalıba özel referans davranışlar (karakteristikler) belirledik. İkinci adımda, uygulamanın karakteristikleri ile referans karakteristikler arasındaki mesafe (euclidean distance) ölçülür ve uygulamanın karakteristiklerine en yakın karakteristiklere sahip referans kalıp uygulamanın kalıbıdır.

Literatürdeki davranışları inceleyerek [5] ve deneyimlerimizi kullanarak referans özellikleri belirledik. Görev paralel ve tekrarlamalı veri kalıplarında, salt okunur ve özel veriler kullanılır çünkü izlekler işlerini verileri paylaşmadan yerel olarak yaparlar. Ardışık düzen ve olay bazlı koordinasyon kalıplarında, safhalar arasında gezici veri kullanılır. Böl ve yönet kalıbında üretici/tüketici ve gezici veriler kullanılır çünkü üretici izlekler, işi alt-işlere böler ve aralarında gezici veriyle haberleşirler. Geometrik ayrıştırma kalıbında, üretici/tüketici şeklinde veri paylaşımı görülür çünkü izlekler, hesaplama sırasında komşu izleklerin verilerine de erişirler.

Görev paralel ve böl ve yönet kalıplarında, izlekler işlerini yerel veriler üzerinde yaptıkları için izlekler arası haberleşme hiç yoktur veya birazdır. Geometrik ayrıştırma ve tekrarlamalı veri kalıplarında, izlekler verilerini komşu izlekler ile paylaştıkları için izlekler arası haberleşme çoktur. Ardışık düzen ve olay bazlı koordinasyon kalıplarında, izlekler arası haberleşme birazdır ve algoritmik olarak tanımlanmıştır.

Ardışık düzen ve olay bazlı koordinasyon kalıplarında, izlekler farklı program sayaçlarına sahiptirler çünkü her izlek ayrı bir safhayı çalıştırır. Diğer kalıplarda ise izlekler aynı program sayaçlarına sahiptir. İş yükünün dengeli olduğu kalıplarda, örneğin geometrik ayrıştırma, izleklerin dinamik komut sayısı ve koşma süreleri benzerken iş yükü dengesiz olduğunda, örneğin böl ve yönet, izlerlerin bu karakteristikleri farklıdır. Görev paralel ve geometrik ayrıştırma kalıplarında, tüm izlekler başlangıçta yaratılırken böl ve yönet kalıbında dinamik olarak koşma sırasında yaratılır.

Örneğin, salt okunur ve özel veri paylaşımı, izlekler arası haberleşmesi hiç olmayan, farklı program sayaçlarına sahip izleklerin, başlangıçta ve aynı izlek tarafından yaratıldığı bir uygulamanın kalıbı görev paraleldir. Çok üretici/tüketici ve biraz gezici veri paylaşımı, izlekler arası çok haberleşmesi olan, aynı program sayacına sahip ve dengeli yüklü izleklere sahip bir uygulamanın kalıbı geometrik ayrıştırmadır.

VI. SENTETİK KARŞILAŞTIRMA UYGULAMASI SENTEZLEME

Bu bölümde, uygulama karakteristiklerini ve paralel mimari kalıpları kullanarak sentetik karşılaştırma uygulamaları geliştirildi. Sentetik karşılaştırma uygulamalarımız, okunabilir ve taşınabilir olan C programlama dilinde yaratılırlar ve MRAPI veya MCAPI kütüphanelerini kullanırlar. Sentetik karşılaştırma uygulamalarımız, gerçek uygulamaların mimari-bağımlı ve bağımsız davranışlarını muhafaza ederler. Bu sayede, yazılımın mimarisel yapısını ve performans karakteristiklerini de korurlar. Aynı zamanda, mevcut uygulama sistemin ortalama performansına sınıyorsa sentetik de ortamala performası sınar ve mevcut uygulama sistemin uç performans değerlerini sınıyorsa sentetik de uç değerleri sınar.

Performans karakteristiklerini korumak için birtakım ölçütler (metrikler) kullandık ve gerçek uygulama ile sentetik karşılaştırma uygulaması arasındaki benzerliği ölçtük. Sentezlemedeki amaç kullanıcının istediği benzerlik seviyesine otomatik olarak erişmektir. Kullandığımız karşılaştırma metrikleri şunlardır: paralel kalıp türü (PL), izlek haberleşmesi (TC), haberleşmenin hesaplamaya oranı (CCR), döngü başına komut sayısı (IPC), önbellek kaçış oranı (CMR) ve hatalı dallanma öngörüsü oranı (BMR). Mikro-mimari bağımlı metriklerin birçoğu daha önceden yapılan benzerlik karşılaştırmalarında kullanılmıştır fakat yazılım tasarım kalıplarını bizden başka kullanan olmamıştır.

Gerçek uygulama ile sentetik arasındaki benzerliği ölçmek için hata oranı hesaplanır. Bir metrik (mt) için gerçek ($mtger$) ve sentetik ($mtsen$) uygulamanın değerleri verildiğinde, *hata oranı (hata yüzdesi)* şu formül kullanılarak hesaplanır: $hataoranı_{mt} = |(mtsen - mtger)/mtger| \times 100$. Daha sonra, hata oranı kullanılarak en düşük değeri 0 ve en yüksek değeri 100 olan benzerlik skoru (benzerlik yüzdesi, $bskor_{mt}$) hesaplanır. Benzerlik skoru, gerçek uygulama ile sentetik karşılaştırma uygulamasının karakteristiklerinin yakınlığını gösterir. Birbirine yakın karakteristikler yüksek benzerlik skoruna, birbirinden uzak karakteristikler düşük benzerlik skoruna sahiptir.

Gerçek uygulama ile sentetik aynı paralel mimari kalıba sahipse, paralel kalıp skoru ($bskor_{pl}$) 100 olur. Paralel tasarım kalıbı gerçek uygulamanın karakteristiklerinin muhafaza edilmesi için önemli olduğundan, sentezleme sırasında $bskor_{pl}$ her zaman 100 olacak şekilde sentezleme yapılır. İzlek haberleşmesi skoru ($bskor_{tc}$), gerçek uygulamadaki ve sentetikteki izlek çiftleri arasındaki haberleşme davranışları karşılaştırılarak hesaplanır. Kalan diğer metrikler için benzerlik skoru, $bskor_{mt} = 100 - hataoranı_{mt}$ şeklinde hesaplanır. Son olarak, *genel benzerlik skoru*, $bskor_{pl}$ hariç diğer skorların ortalaması alınarak hesaplanır.

Sentezlemede ilk olarak gerçek uygulamanın paralel kalıbı ve karakteristikleri kullanılarak sentetik karşılaştırma uygulaması yaratılır. Yaratılan sentetik, gerçek uygulama ile aynı sayıda izleğe sahiptir ve izleker arasındaki haberleşme gerçek uygulamadakine göre oluşturulur. İzlekler arası haberleşme (MRAPI kullanıldığında) okuma/yazma veya (MCAPI kullanıldığında) mesaj gönderme/alma şeklinde olur. Yaratılan ilk sentetik, paralel kalıp için belirlenen referans davranışları taşıyacak şekilde sentezlenir. Örneğin, görev paralel kalıbında, veri paylaşım davranışının özel olması için yerel veriler oluşturur ve bunlar üzerinde okuma/yazma işlemleri yapılır. Benzer şekilde, veri paylaşım davranışının salt okunur olması için global veri yaratılıp bunun üzerinde işlemler yapılır. Bu kalıpta, izlekler arası haberleşme olmadığı için haberleşme eklenmez ve izleklerin program sayaçları farklı olduğu için her izlek farklı bir fonksiyon çalıştırır. Koşma süreleri aynı olması için izlekler aynı oranda hesaplama yaparlar.

İlk sentetik karşılaştırma uygulaması yaratıldıktan sonra genel benzerlik skoru hesaplanır ve (gerekliyse) bu skor iterasyonlarla arttırılmak üzere yeni sentetikler yaratılır. Eğer kullanıcı tarafından belirlenen genel benzerlik skoruna veya iterasyon sayısına erişilmişse, sentezleme işlemi tamamlanır. Aksi halde hedeflenenden düşük benzerlik skoruna sahip metrikler bulunur ve yeni sentetik yaratılarak bu skorlar iyileştirilir. Örneğin, $bskor_{tc}$ düşük ise, gerçek uygulamadaki izlekler arasında olup da sentetikte olmayan haberleşmeler, sentetiği de eklenir veya gerçek uygulamada olmayıp da sentetikte olan fazla haberleşmeler kaldırılır. $bskor_{ipc}$ düşük ise, yüksek (toplama işlemi) veya düşük (bölme işlemi) IPC değerine sahip bir C kod bloğu eklenir. $bskor_{cmr}$ düşük ise, önbellekte olan veriye çok defa erişilen veya önbellekte olmayan bir veriye erişilen bir C kod bloğu eklenir. Benzer şekilde diğer metrikler için de kod blokları eklenir.

Tablo I
PARSEC UYGULAMALARININ PARALEL KALIP TANIMLAMA VE SENTEZLEME SONUÇLARI

Gerçek			Sentetik		
Uygulama	**SS**	**Paralel Kalıp**	**SS**	**Paralel Kalıp**	**İS**
Blackscholes	1262	Görev Paralel	116	Görev Paralel	1
Bodytrack	7696	Geo. Ayrıştırma	1197	Geo. Ayrıştırma	5
Canneal	2794	Görev Paralel	116	Görev Paralel	1
Dedup	7125	Ardışık Düzen	756	Ardışık Düzen	1
Facesim	20275	Görev Paralel	190	Görev Paralel	1
Ferret	10765	Ardışık Düzen	2722	Ardışık Düzen	4
Fluidanimate	2784	Geo. Ayrıştırma	867	Geo. Ayrıştırma	9
Swaptions	1095	Görev Paralel	189	Görev Paralel	6
X264	38546	Ardışık Düzen	1647	Ardışık Düzen	17

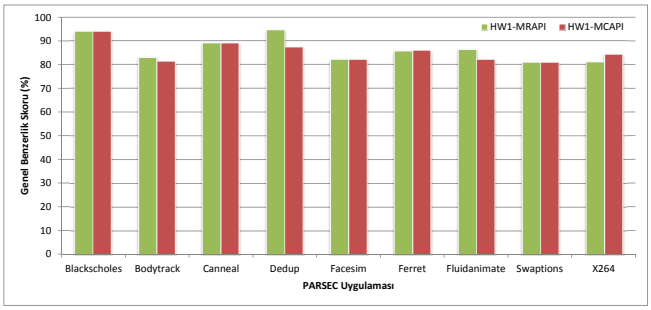
Yukarıda anlatılan tüm adımlar tamamlandığında, çok çekirdekli sistemler için minyatür bir sentetik karşılaştırma uygulaması elde edilir. Deneylerde gösterileceği gibi elde edilen sentetik karşılaştırma uygulaması, gerçek uygulamanın performans karakteristiklerini taşımaktadır.

VII. DENEYLER

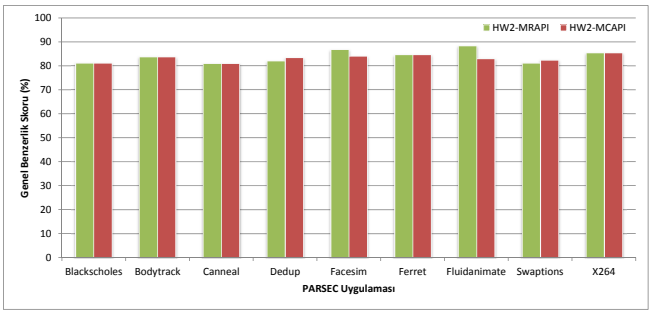
Gerçek uygulamalar ile onlar için yaratılan sentetikler arasındaki benzerlikleri analiz etmek için deneyler yaptık. Tekniğimizin farklı sistemlerde çalıştığını göstermek için farklı çekirdek sayılarına ve önbellek boyutlarına sahip sistemler kullandık. Deneyleri iki farklı donanımda (sistemde) yaptık. HW1 sistemi, 4 çekirdekli, 6 MB önbellekli i7 işlemciye ve HW2 sistemi, 2 tane 8 çekirdekli, 8 MB önbellekli Xeon e5520 işlemciye sahiptir. Gelecekte, gömülü sistemlerde yaygın olarak kullanılan farklı sistemler üzerinde deneylerin genişletilmesi planlanmaktadır. PARSEC [1] uygulamalarından MRAPI ve MCAPI kütüphanelerini kullanan sentetik karşılaştırma uygulamaları geliştirdik. Karakteristikleri toplamak için dinamik ikili enstrümantasyon aracı olan DynamoRIO [9] ve bellek gölgeleme aracı olan Umbra'yı [10] kullanarak yeni bir araç geliştirdik. Ayrıca mikro-mimari bağımlı karakteristikleri toplamak için (Linux) perf aracını [11] kullandık. Deneylerde, genel benzerlik skoru olarak %80 ve iterasyon sayısı üst limiti olarak 20 kullanıldı. İlk mimari ve performans analizlerinde, bu benzerlik skoru yeterli olabilir ancak son performans değerleri için sadece sentetik karşılaştırma uygulamaları kullanılmamalıdır.

Tablo I, kalıp tanımlama sonuçlarımızla birlikte PARSEC uygulamalarının literatürde bilinen kalıplarını göstermektedir. Tüm uygulamaların paralel kalıpları doğru şekilde bulundu. Tabloda, gerçek ve sentetik uygulama için kod satır sayıları (SS) ve sentezlemenin kaç iterasyonda yapıldığı (İS) verilmektedir. Görüldüğü gibi sentetikler daha küçük ve daha az karmaşıktır dolayısıyla yüksek hızda simülasyona olanak sağlarlar. Bütün deneylerde, sentezleme birkaç iterasyonda tamamlandı. `X264`'ün sentetiği büyük olduğu ve içerisinde çok sayıda (MCAPI/MRAPI) kütüphane fonksiyonu çağrısı olduğu için 17 iterasyonda tamamlandı çünkü çok sayıdaki çağrının, benzetmeye çalıştığımız metrikler üzerinde büyük etkisi vardır.

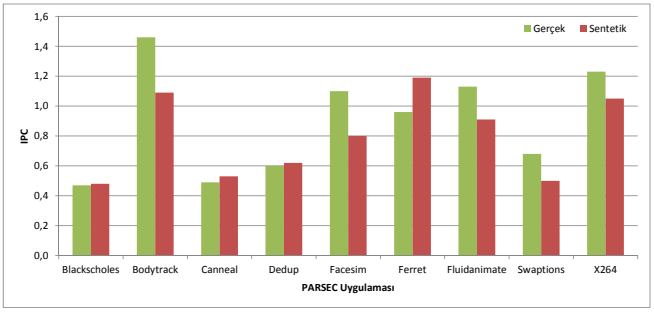
Daha yüksek benzerlik oranları talep edildiğinde, iterasyon sayısındaki değişimi görmek için bazı deneyler



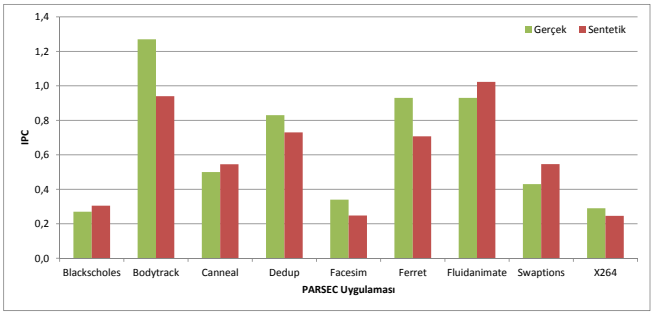
Şekil 3. HW1 üzerinde, PARSEC MRAPI/MCAPI sentetikleri için genel benzerlik oranları



Şekil 5. HW2 üzerinde, PARSEC MRAPI/MCAPI sentetikleri için genel benzerlik oranları



Şekil 4. HW1 üzerinde, PARSEC uygulamaları ile MRAPI sentetiklerinin IPC değerlerinin karşılaştırılması



Şekil 6. HW2 üzerinde, PARSEC uygulamaları ile MRAPI sentetiklerinin IPC değerlerinin karşılaştırılması

yaptık. Bu deneylerde, iterasyon sayısının bazı durumlarda aynı kaldığını bazı durumlarda ise lineer olarak yükseldiği gözlemledik. Örneğin, genel benzerlik skoru olarak %90 seçildiğinde, `Blackscholes` için yine 1 iterasyonda sentetik yaratılmaktadır. `Bodytrack` için ise iterasyon sayısının 5'ten 8'e yükseldiği görüldü. Ek olarak, benzerlik skorunun yükselmesinin, sentetiğin satır sayısı üzerinde etkisi olmadığını görüldü.

Kalıp tanımlama sonuçlarından sonra mikro-mimari bağımlı ve bağımsız metrikler kullanarak gerçek ve sentetik uygulamalar arasındaki benzerliği ölçtük. Şekil 3, HW1 üzerinde MRAPI ve MCAPI sentetiklerinin genel benzerlik skorunu gösteriyor. MRAPI için ortalama genel benzerlik skoru %86, minimum skor (`Swaptions` ve `X264`'de) %81 ve maksimum skor (`Dedup`'da) %95'tir. MCAPI için ortalama genel benzerlik skoru %85, minimum skor (`Bodytrack` ve `Swaptions`'da) %81 ve maksimum skor (`Blackshole`s'da) %94'tür. Beklendiği gibi, genel benzerlik skoru kullanıcı tarafından tanımlanan orandan (%80) yüksektir.

Şekil 4, HW1 üzerinde gerçek ve sentetik uygulamanın IPC değerlerinin karşılaştırma sonuçlarını gösteriyor. Ortalama hata oranı %17 ve maksimum hata oranı (`Facesim` ve `Swaptions`'da) %27'dir. Yer yokluğu nedeniyle, MCAPI sentetikleri ve diğer metrikler için sonuçları veremiyoruz ancak onların da %80'nin üzerinde benzer olduğu gözlemlendi [12].

Benzerlik deneylerinden sonra sentetiklerin donanımdan bağımsızlığını görmek için deneyler yaptık. HW1 üzerinde sentezlenen sentetikler, tekrar sentezlenmeden, HW2 üzerinde koşturuldu. Şekil 5, HW2 üzerinde MRAPI ve MCAPI sentetiklerinin genel benzerlik skorunu gösteriyor. MRAPI için ortalama genel benzerlik skoru %85, minimum skor (`Swaptions` ve `X264`'de) %81 ve maksimum skor (`Dedup`'da) %93'tür. MCAPI için ortalama genel benzerlik skoru %84, minimum skor (`Swaptions`'da) %81 ve maksimum skor (`Blackscholes`'da) %86'dır.

Şekil 6, HW2 üzerinde gerçek uygulamanın ve HW1 üzerinde sentezlenen sentetik uygulamanın IPC değerlerinin karşılaştırma sonuçlarını gösteriyor. Ortalama hata oranı %18 ve maksimum hata oranı (`Facesim` ve `Swaptions`'da) %27'dir. Bu sonuçlara göre genel benzerlik skoru da IPC skoru da donanımdan (sistemden) bağımsızdır çünkü sistem değiştiğinde neredeyse aynı benzerlik skorlarını gözlemledik. Yer yokluğu nedeniyle, MCAPI sentetikleri ve diğer metrikler için HW2 üzerinde sonuçları veremiyoruz ancak onların da benzerlik oranlarını korudukları gözlemlendi [12].

VIII. SONUÇLAR

Bu çalışmada, çok çekirdekli sistemler için sentetik karşılaştırma uygulamaları sentezleyen otomatik bir araç geliştirdik. Uygulamaların karakterizasyonu için yazılım mimari kalıplarından faydalandık ve kalıpların uygulamaların üst seviye karakteristiklerini elde etmek için önemli olduğunu gördük. MRAPI ve MCAPI kütüphaneleri sayesinde, geliştirdiğimiz sentetikler altyapıdan (SMP, mesaj iletim mimarileri, vb.) bağımsız olarak çok çekirdekli heterojen sistemlerde koşabilirler. Aynı zamanda, daha önceki sentetiklerden farklı olarak C programlama dilinde olan sentetiklerin okunabilirlikleri yüksektir. Sentetiklerimizin gerçek uygulamalarla benzer karakteristiklere sahip

olduklarını deneysel olarak doğruladık.

IX. TEŞEKKÜR

Bu çalışma Semiconductor Research Corporation 2082, Boğaziçi Üniversitesi BAP 7223 ve Türkiye Bilimler Akademisi GEBİP kapsamında desteklenmiştir.

X. KAYNAKÇA

[1] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[2] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009.

[3] "Embedded Microprocessor Benchmark Consortium, http://www.eembc.org," 2014.

[4] "Multicore Association, http://www.multicore-association.org," 2014.

[5] T. Mattson, B. Sanders, and B. Massingill, *Patterns for parallel programming*. Addison-Wesley, 2005.

[6] J. L. Ortega-Arjona and G. Roberts, "Architectural patterns for parallel programming," in *European Conference on Pattern Languages of Programs (EuroPLoP)*, 1998.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[8] K. Keutzer, B. L. Massingill, T. G. Mattson, and B. A. Sanders, "A design pattern language for engineering (parallel) software: merging the plpp and opl projects," in *Proceedings of the 2010 Workshop on Parallel Programming Patterns (ParaPLoP)*, 2010.

[9] "DynamoRIO Dynamic Instrumentation Tool Platform, http://dynamorio.org/," 2014.

[10] Q. Zhao, D. Bruening, and S. Amarasinghe, "Umbra: Efficient and scalable memory shadowing," in *The International Symposium on Code Generation and Optimization (CGO)*, 2010.

[11] "Linux profiling with performance counters, https://perf.wiki.kernel.org," 2014.

[12] E. Deniz, A. Sen, J. Holt, and B. Kahne, "Using Software Architectural Patterns for Synthetic Embedded Multicore Benchmark Development," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2012.