Çok İplikli Programlarda İplik Davranışlarının İlgiye Yönelik Programlama Yaklaşımıyla Analizi

Analysis Of Thread Behaviour Using Aspect-Oriented Programming Approach in Multi-Threaded Programs

Oral, Alan
TÜBİTAK Ulusal Elektronik ve Kriptoloji Araştırma Enstitüsü
Gebze,Kocaeli
oral.alan@bte.tubitak.gov.tr

Nadia, Erdoğan
Bilgisayar Mühendisliği Bölümü
İstanbul Teknik Üniversitesi,
İstanbul
nerdogan@itu.edu.tr

Özet

Çok iplikli programlarda iplik davranışlarının anlaşılması, çok iplikli program geliştirmeyi kolaylaştırmaktadır. İpliklerin durum geçişleri ve durum zamanları üzerinden hesaplanan metriklerle davranışları anlamlı hale getirilebilir. Bu çalışmada ilgiye yönelik programlama yaklaşımıyla Java programlama dilinde iplik davranışlarını analiz etmek üzere ilgiye yönelik bir iplik izleyicisi geliştirilmiş ve kullanımı bir üretici-tüketici problemi test uygulaması üzerinden gösterilmiştir. İlgiye yönelik programlama sistemin genelini ilgilendiren kavramları sistem parçalarından ayrı olarak ele alır. İlgiye yönelik programlamanın iplik izleme ve metrik hesaplamada verimli bir yöntem olup olmadığı test sonuçları ve ek izleme yükünün diğer izleyicilerle karşılaştırılmasıyla tartışılmıştır.

Abstract

Understanding thread behaviour makes multi-thread programming easier in multi-threaded programs. The behaviour will be made understandable by calculating the metrics that are driven from thread state change and the times spent on that states. In this work, we present a thread profiler using aspect oriented programming approach and the usage of the profiler on a producer-consumer test application. AOP handles the general concepts seperate from the components of a system. The conclusion whether thread profiling and metric calculations using AOP is efficient is presented using test results and overhead comparison with other profilers.

1. Giriş

Çok-iplikli(multi-thread) programlar geliştirilirken en önemli sorun ipliklerin (thread) gerçek sistemde nasıl davranacağının önceden belirlenememesi ve ipliklerin çalışması hakkında iplik bazında detaylı bilgiye sahip olunamamasıdır. Metrik tabanlı yöntemler doğru olay ve zaman bilgisinin toplandığı durumlarda karmaşık sistemlerin çalışma durumları ve başarımları hakkında bilgi vermektedir.

Bu çalışmada iplik davranışları metrikler ile analiz edilmiş ve metrikleri elde etmede kullanılmak üzere JAVA sanal makinesi üzerinde ilgiye yönelik programlama yaklaşımıyla geliştirilen iplik izleyici anlatılmıştır. JAVA uygulama geliştirme ortamında hassas ölçüm yapabilmek için işlemci saat frekansı ve tik sayısının elde edilmesi gereklidir.

2. İplikler ve Çok İplikli Sistemler

İplik (thread), işlemci kullanımının küçük ve basit bir birimidir ve program içerisinde mantıksal akışa karşılık gelir. Bir iplik gerçekleme bakımından yığın (stack), kütük seti (register set) ve program sayacının oluşturduğu yapıdır. İplikler işlemlerin (process) içinde yer alırlar. İşlemler kaynak kullanımı bakımından ipliklerden daha büyük birimlerdir. Geleneksel yapıda bir işlem tek ipliğin denetimindedir ve birim zamanda tek iş yapabilir. Bu tarz işlemlere tek iplikli işlemler adı verilir. Bir işlemde birden fazla ipliğin olduğu yapılar ise çok iplikli işlemler olarak adlandırılır. Çok iplikli işlemler işleme ayrılmış olan kaynakları paylaşırlar ve her ipliğin belli görev ya da görevleri üstlenmesi sayesinde, işlemler birden fazla görevi aynı anda gerçekleştirebilir hale gelirler. Günümüzde simetrik çok işlemcili bilgisayarların yaygınlaşmasıyla ipliklerin bu sistemlerde verimli şekilde çalıştırabilmesi önem kazanmıştır. Doğru şekilde eş zamanlı parçalara ayrılmamış ve senkronizasyonu sağlanmamış programlarda başarım kolaylıkla sıralı şekilde yazılmış programların başarımının altına düşebilmektedir. Bu nedenle geliştiriciler ipliklerin davranışları ve

durumları hakkında bilgi sahibi olmalı ve testlerle başarımı ölçmelidirler.

Çok iplikli sistemler çalışma sırasında öngörülemeyen karmaşık davranışlar sergilerler. Yazılım tasarımı sırasında yazılım mimarları ve geliştiriciler mantıksal ayrıştırma sürecinden sonra gerçek sistemde ipliklerin davranışlarının nasıl olacağı sorusunu cevaplamaya çalışırlar. İpliklerin çalışması iplik önceliği, işlemci çekirdeği seçimi, çalışma durumu (başladı, çalışıyor, bekliyor) ve sistemin gecikmesi gibi birçok etkene bağlıdır.[1]

Davranış tahminlerine göre ipliklerin senkronizasyonu ve yönetimi biçimlenir. Ancak ne kadar iyi tasarlanmış olursa olsun ipliklerin davranışlarını anlayabilmek için ipliklerin gerçek sistemdeki çalışma durumları hakkında bilgi edinilmelidir.

2.1. Çok İplikli Programların Davranışlarını Anlama Yaklaşımları

Programların davranışlarını anlama yaklaşımları genel olarak statik ve dinamik analizler olmak üzere ikiye ayrılmaktadır. Çok iplikli programlar da benzer analiz yöntemlerinden faydalanılarak analiz edilir. Statik yaklaşım, programın çalışmasına gerek duymaz ve model üzerinden beklenen davranışın incelenmesine dayanır. Bu yaklaşımda statik metrikler üzerinden davranış için fikir edinilmeye çalışılır. Bu metriklere kod satır sayısı, tasarım karmaşıklığı gibi metrikler örnek olarak verilebilir.

İplik davranışlarını anlamada genellikle kullanılan yöntem dinamik analizdir. Dinamik analiz izleme olarak da isimlendirilir. İzlemenin avantajı direkt olarak sistemin çalışması esnasında elde edilen verilere dayanması ve sistemin parçalarını çalışma zamanlı ele alabilmesidir. İzlemeden elde edilen verilerden dinamik metrikler türetilebilir [2]. Bu metrikler temelde sistemden elde edilen zaman ve olay bilgisine dayanır. İpliklerin davranışlarında kullanılan metrikler aşağıda sıralanmıştır.

- **Cevap süresi:** İpliğin yaratılmasından sonlanmasına kadar geçen toplam süre.
- **Kullanım oranı:** İpliğin çalışıyor durumunda bulunduğu zamanın cevap süresine oranı.
- **Kritik yol:** İpliğin bir durumda bulunduğu en uzun süre.

2.2. Çok İplikli Sistemleri İzlemenin Zorlukları

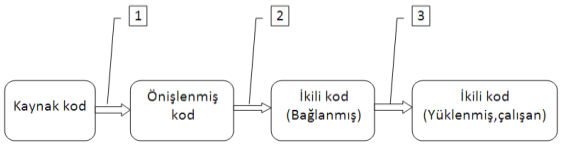
Çok iplikli sistemleri izlemek, sistemlerin ve ipliklerin bazı karakteristik özelliklerinden dolayı zorluklar taşır. [1]

- **Ölçülecek olayların yüksek sıklıkta gerçekleşmesi.** İplikler arası olaylar ve durum geçişleri sıklıkla gerçekleşir. Ölçülen sistemdeki ölçme için sisteme getirilecek ek yük dikkatli şekilde incelenmelidir. Eğer ölçme yükü fazla olursa sistemin davranışı bundan etkilenir ve gerçekçi sonuçlar elde edilemez.
- **Katmanlar arası eşleştirme.** İplikler sistemlerde birçok katmanda yer alabilirler. Örneğin işletim sisteminde, sanal makinede veya uygulamanın içinde iplikler kullanılabilir. Katmanlar arası iplik eşleştirmesi ve bu eşleşmenin gösterilebilmesi, iplikler arası eşleştirmeler bire-bir olmadığından ve dinamik olarak sağlandığından çok zordur.
- **Karmaşık ilişkilerin izlenmesi.** İplikler yürütmenin temel birimleridir ve genellikle bağımsızdırlar. İplikler arası etkileşim bellek, dosya ve diğer araçlar gibi sistem kaynakları üzerinden sağlanır. Bu sistem kaynaklarının izlenmesi çok fazla ek izleme yükü getireceğinden dolayı genellikle uygulanmaz.

Çok iplikleri programları izlerken izlemenin zorluklarından dolayı uygulamalar her izlemede farklı sonuçlar verecek şekilde çalışabilir. İzleme sırasında işletim sisteminde iplik değişiminden dolayı gecikmeler olabilir.

3. Çok İplikli Sistemlerin Davranışını Anlamada Kullanılan Yöntemler

Programın yazılmasından koşturulmasına kadar geçen adımlarda davranış analizine yönelik işlemler yapılabilir. Bu işlemlerin amacı o aşamada izlemeyle ilgili bilgi elde edilmesidir. Şekil 1'de izleme ile ilgili ekleme yapılabilecek aşamalar gösterilmiştir.[1]



Şekil 1: Derleme Esnasında İzleme Bilgisi Eklenebilecek Adımlar

3.1. Derleyici-Tabanlı İzleme Yöntemleri

Şekil 1'de gösterilen 1 ve 2 numaralı adımlarda gerçekleştirilebilecek yöntemlerdir.

3.1.1. Kaynaktan-Kaynağa Dönüşüm

Büyük sistemlerde bütün kaynak kodu izleme için değiştirmek oldukça zordur. Bunun için kaynak kodda ilgili noktaları belirleyip bu noktalara izleme bilgisi için kod ekleyebilecek ön bilgiye dayanan geliştirme çatıları ortaya çıkmıştır. Dönüştürülen programlar yeniden derlenir ve bu sayede izleme bilgisi ikili koda eklenmiş olur. Bu yönteme örnek olarak Proteus[3] verilebilir. İlgiye yönelik programlama[4] da kaynaktan kaynağa dönüşümün bir örneğidir.

3.1.2. Statik İkili Kod Dönüşümü

İzleme bilgisi eklemenin bir diğer yolu derlenmiş ikili koda ekleme yapmaktır. Birçok derleme ve izleme aracı (Rational Purify and Quantify[5]) kaynak kodu izleme bilgisi ekleyerek ve önceden hazırlanmış kütüphanelere bağlayarak derleyebilmekte ve böylece izlemeyi sağlayabilmektedir.

3.1.3. Dinamik İkili Kod Dönüşümü

İzleme bilgisi eklemenin en zor ve alt seviye yolu dinamik ikili kod dönüşümüdür. Şekil 1 'de 3 numara ile bağlanmış ikili kodla, yüklenmiş ve çalışan ikili kod aşamaları arasında yer alır. Bu yöntem Just-in-Time(JIT) derlemeyi kullanır. Avantajı izleme işleminin program derlenmeden eklenebilmesi ve sadece ihtiyaç duyulan zamanda kullanılabilmesidir. İzleme istenmeyen durumda devreden çıkartılır ve bu sayede izleme ek yükü azaltılmış olur.

Paradyn [6] paralel programların başarımını ölçmek ve programlardaki başarımı azaltan nedenleri bulmak için geliştirilmiş bir sistemdir. İzleme bilgisi toplanan düğüm için, başarımı azaltan bir neden olabileceği düşünülürse, onun altında yer alan kaynaklar ve sorunları değerlendirmek üzere tekrar izleme bilgisi toplanır ve işlem bittikten sonra izleme bilgisi toplama işlemine son verilir. Bu şekilde oluşturulmuş hipotezler üzerinden paralel sistemler taranarak sorunları belirlenmeye çalışılır.

3.2. İşletim Sistemi ve Arakatman İzleme

Bütün uygulamalar işletim sisteminin sunduğu servisleri kullanırlar. Bu servisler paylaşılan kaynaklara erişimi sağlarlar. İplikler bir sistem çağrısı yaptığı anda sistem çekirdek moduna geçer. Bu geçiş noktası uygun yapılarla ele alınıp, iplik bazında kaynak kullanımının ölçüldüğü izlemedir.

3.3. Sanal Makine İzleme

Sanal makinelerin kullanımı iplikleri izlemeyi kolaylaştırır. Sanal makine kullanıcı programıyla işletim sistemi arasında yer alır. Uygulamalar sanal makine bağlamında çalışırlar ve bu sayede arakoda erişmek ve izlemek daha kolaydır. Daha önce bahsedilen izleme yöntemleri sanal makineler için de geçerlidir.

3.3.1. Arakod Dönüşümü

Arakod (bytecode) dönüşümü ikili kodda olduğu gibi program arakoduna izleme bilgisinin eklenmesine dayanır.

4. İlgiye Yönelik Programlama

İlgiye yönelik programlama (aspect oriented programming), nesneye yönelik programlama yaklaşımında eksik kalan özellikleri tamamlama, kolay yönetilebilen ve sistemde olması gereken kavramların işlevsel olarak ayrıldığı bir yaklaşım olarak ortaya çıkmıştır. Bu yaklaşımda kullanılan temel kavramlar şunlardır [7].

- **İlgi.** Sistemin temel işlevlerini yerine getirmesini sağlayan ve sistemin gerçekleştirmesi gereken işlevsel parçalara karşılık gelen kavramdır.
- **Enine kesen ilgi.** Sistemin bütün ya da belirli parçalarını etkileyen ilgilerdir.
- **Birleşme noktaları**. Bir enine kesen ilgiyle ilgili işlemlerin başlaması gereken yerlerdir.
- **İcra noktaları.** Enine kesen ilgide belirtilen kodun hangi metodda çalışmaya başlayacağının tanımlandığı noktalardır. Metodun çalışmaya başlamasıyla ilgili 3 farklı icra noktası tanımı mevcuttur.

  Öncesinde. Metodun birleşme noktasına gelinmeden hemen önce çalışacağını belirtir.

  Esnasında. Birleşme noktası olarak belirtilen metodun yerine çalışması gereken metodu belirtir.

  Sonrasında. Belirtilen metodun, birleşme noktasında çalışması gereken metoddan sonra çalışacağını belirtir.

- **Dokuma.** Belirtilen ilgilerin ve sistem parçalarının nasıl biraraya getirileceğinin belirtilmesidir.

  Derleme-zamanı. Uygulama kodu ve ilgiler uygulama derlenmeden önce biraraya getirilip birlikte derlenirler.

  Birleşme-zamanı. Uygulama ve ilgiler ayrı ayrı arakoda derlendikten sonra dokunurlar.

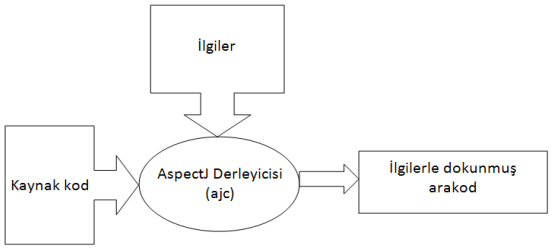
Yükleme-zamanı. Ayrı ayrı arakoda çevrilen uygulama ve ilgilerin, sınıf yükleyicisi tarafından yüklenmesidir.

Çalışma-zamanı. Sanal makine, birleştirme noktalarını tespit eder ve ilgili metodu yeri geldiğinde çalıştırır.

İlgiye yönelik programlamada sistemin bütününü etkileyen parçaların ayrılması ve bu parçaların ilgiler olarak gerçeklenip uygulamayla birlikte dokunması, ilgiye yönelik programlamayı izleme işlemi için en uygun yöntemlerden biri haline getirmiştir. Bu çalışmada Java'da ipliklerin izlenebilmesi için ilgiye yönelik programlama ile bir iplik izleyicisi geliştirilmiştir. Java'nın eş zamanlı program geliştirme arayüzü üzerinde gerçeklenen ilgiler ile iplik geçişleri ve zaman bilgileri iplik bazında kaydedilmiştir. Kaydedilen bu değerlerden iplik analizinde kullanılan toplam cevap süresi, kullanım oranı ve kritik yol metrikleri elde edilmiştir.

4.1. AspectJ

İlgiye yönelik programlama yaklaşımının programlama dillerine uyarlanmış bir çok gerçekleştirimi mevcuttur.



Şekil 2: AspectJ Derleme-Zamanlı Dokuma

AspectJ[4], Java programlama diline ilgiye yönelik programlama yeteneklerini kazandıran ve yaygın kullanılan bir ilgiye yönelik programlama gerçekleştirimidir. İlgileri uygulamalarla beraber derleyebilmek için ajc (aspect java compiler) isimli derleyiciyi kullanmaktadır. Şekil 2' de AspectJ'in derleme-zamanlı kullanımı gösterilmektedir.

Diğer yaygın gerçeklemeler Jboss AOP[8], AspectWerkz[9] ve Spring AOP[10] dir.

Tablo 1'de AspectJ ve diğer gerçeklemelerin, öncesinde ve esnasında icra noktalarının her çağrımı başına geçen nanosaniye cinsinden getirdikleri ek izleme yükleri karşılaştırması verilmiştir [11].

Tablo 1: İYP gerçekleştirmeleri karşılaştırması

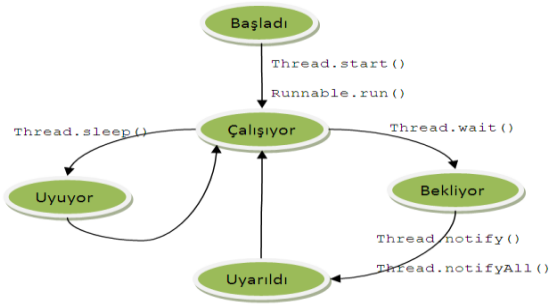
İYP Karşılaştırma	Aspect werkz	Aspec tJ	Jboss AOP	Spring AOP
Öncesinde, args() target()	10	10	220	355
Esnasında x 2, args() target()	80	50	290	436

Karşılarştırma verilerinden AspectJ 'in en az ek yük getirdiği anlaşılmaktadır. Bu çalışmada ek izleme yükünü azaltmak için AspectJ gerçekleştirimi kullanılmıştır.

5. Java İplikleri

Diğer bütün güncel programlama dillerinde olduğu gibi JAVA programlama dilinde de iplikler kullanılır. JAVA iplikleri programlayabilmek için programcılara bir uygulama geliştirme arayüzü sunmaktadır.

İpliklerin davranışlarını anlayabilmek için ipliklerin bulunabileceği durumlar belirlenmeli ve bu durumlar arasında zaman bilgisi toplanmalıdır. Bu sayede iplik durum-zaman bilgisi toplanabilecek ve dinamik metriklerin iplik bazında hesaplanabilmesi sağlanabilecektir.



Şekil 3: İplik Durumları ve Geçişleri

Şekil 3'de bu çalışmada kullanılan ve Java programlama dilindeki ipliklerin bulunabileceği durumlar ve bu durumlar arasında geçişlerin hangi çağrılarla yapıldığı gösterilmiştir. Bu çağrılar ilgiye-yönelik programlamada birleşme noktaları olarak ele alınabilecek yerleri göstermektedir.

6. Java İlgiye Yönelik İplik İzleyici (JİYİPİZ)

Bu bölümde geliştirilen iplik izleyicisinin mimarisi ve iplik izleme ilgisinin yapısıyla ilgili bilgi verilmektedir. Ayrıca JİYİPİZ'in izleme için nasıl kullanılacağı anlatılmıştır.

6.1. JİYİPİZ Mimarisi

Java programlama dilinin sunduğu uygulama geliştirme arayüzü üzerine iplik durumlarını ve geçişlerini nano saniye bazında kaydedebilen Java İlgiye Yönelik İplik

İzleyicisi (JİYİPİZ) geliştirilmiştir. Şekil 4'de JİYİPİZ'in genel mimarisi gösterilmektedir.

Şekil 4: JİYİPİZ Genel Mimarisi

JİYİPİZ' de Şekil 3'de gösterilen birleşme noktalarını yakalayan ve bu noktalarda iplik denetçisini çağıran bir iplik izleyici ilgisi bulunmaktadır.

Denetçi işlemlerini kritik bölgeler içinde gerçekleştirmektedir. Böylece bir iplik yakalandıktan sonra denetçi tarafından kaydı yapılana kadar başka bir ipliğin çalışmasına izin verilmez. Çağırılan iplik denetçisi iplikle ilgili kaydı iplik veri tutucusuna yapmaktadır. İplik veri tutucusu, ipliklerin numarası, geçtiği durumu ve anlık saat bilgisini alarak veri yapısına ekler. Ayrıca bu ekleme işlemi sırasında bir önceki tutucunun zaman bilgisini yeni aldığı saat bilgisinden çıkartarak bir önceki ipliğin o durumda ne kadar kaldığını kaydeder. İplikler sonlandıktan sonra iplik denetçisi iplik veri tutucusu üzerinde kaydedilen bilgilerden iplik bazında metriklerin hesaplama işlemini yapmak üzere başarım metrik hesaplayıcısını çağırır ve iplik bazında metrikler hesaplanarak dosyaya kaydedilir.

Şekil 3'de belirtilen iplik durum geçişlerini yakalamak üzere kullanılan birleşme noktaları tanımlamaları Tablo 2'de gösterilmiştir.

Tablo 2: İplik izleyici ilgisinde kullanılan birleşme noktaları ve ifadeleri

Tanım	İcra Noktası	İcra Noktası İfadesi
İplik Başlangıç Çağrısı	start()	call
Runnable Arayüz Başlangıç Çağrısı	Runnable+.run()	call
İplik Bekleme Çağrısı	wait()	call
İplik Uyku Çağrısı	sleep()	call
İplik Uyarma Çağrısı	notify()	call
Bütün İplikleri Uyarma Çağrısı	notifyAll()	call

Bu birleşme noktalarında yakalanan çağrılarda durum geçişlerin gerçekleşmeden öncesinde icra noktası kullanılarak ipliklerin durum geçişleri kaydedilmiş ve durum değişikliği sonlanmış ancak başka duruma geçmemiş iplikler içinde sonrasında icra noktası ile zaman bilgileri toplanarak kaydedilmiştir.

6.2. JAVA Sanal Makinesinden İşlemci Zamanı Bilgisi Toplama

İpliklerle çalışırken yöntemin doğası gereği iplikler çok hızlı şekilde durum değiştirebilmektedir. JİYİPİZ ve metrik tabanlı analiz yöntemi bütünüyle zaman bilgisi üzerine kuruludur. JAVA uygulama geliştirme platformu yüksek seviye olarak nitelendirilen bir ortamdır. Bu ortamda yapılan testlerde iplikler arası geçişleri görebilmek için milisaniye bazında ölçümler dahi yetersiz kalabilmekte ve bu nedenle bütün durumları kapsayabilecek hassasiyette bir değerde ölçüm yapılması gerekmektedir. Bu nedenle JİYİPİZ nanosaniye düzeyinde ölçümleri desteklemektedir. Bu düzeyde ölçüm yapabilmek için JAVA platformundan en alt seviyeye inerek işlemcinin anlık tik sayısına erişim ihtiyacı doğmuştur. JAVA işlemciye direk erişim gibi özellikleri doğal arayüz (native interface) yardımıyla sunmaktadır.

İplik geçişlerinde doğru zaman bilgisini elde edebilmek için JİYİPİZ'de kullanılan çalışma ortamında derlenmiş bir paylaşılan kütüphane (shared library) kullanılmıştır. Bu kütüphane yardımıyla JİYİPİZ, C programlama dilindeki yordamları çağırmakta ve bu yordamlar da yardımcı (assembly) dil aracılığıyla ipliğin çalıştığı işlemciden işlemci numarası ve anlık tik (tick) sayısını almaktadır. Anlık tik sayısı birçok genel amaçlı işlemci tarafından desteklenen 'read time-stamp counter' (rdtsc) [12] komutu, işlemci numarası da 'cpuid' [13] komutu kullanılarak alınmıştır.

Ayrıca tik sayısından geçen zamanı hesaplayabilmek için işlemcinin saat frekansının da bilinmesi gerekmektedir. JİYİPİZ, çalışmaya başladığında yine yardımcı kütüphane aracılığıyla işlemci frekansını da almakta ve 1'e göre iki iplik durumu arasındaki geçen zamanı hesaplamakta ve veri yapısında ilgili yere kaydetmektedir.

$$\#\text{ nano saniye} = \#\text{ tik farkı} / \text{frekans} * 1000 \quad (1)$$

Burada frekans MHz cinsinden alınmıştır. (1 Mhz = 1.000.000 Hz). Elde edilecek sonuçlarda mümkün olan en hassas ölçüm elde edilmek istendiğinden sonuç nanosaniyeye çevrilmiştir.

6.3. JİYİPİZ'in Kullanımı

JİYİPİZ derleme zamanında dokunacak şekilde tasarlanmıştır. JİYİPİZ ipliklerin kullanıldığı Java programlarında projeye bir kütüphane olarak eklenmelidir. Projeye eklendikten sonra projede yer alan sınıfları dokuyarak, ilgide yer alan birleşme noktalarını bulur ve derlendiğinde izleme bilgisini arakoda ekler. Program çalıştırıldıktan sonra tüm ipliklerin çalışması bittikten sonra JİYİPİZ'in hesapladığı sonuçlar çıktı olarak Eclipse geliştirme ortamına eklenmiş uyumlu-ek (plug-in) ile gözlemlenebilir.

7. JİYİPİZ'in Uygulama Üzerinde Kullanılması

Bu bölümde üretici-tüketici problemi üzerinde JİYİPİZ kullanılarak iplikleri izleme çalışması anlatılmıştır.

7.1. Üretici-Tüketici Problemi

Üretici-tüketici problemi paralel çalışmanın klasik problemlerinden biridir. Ortak kullanılan bir tampon üzerinde üretici iplikler çalıştığında tampona ekleme, tüketiciler ise çalıştıklarında tampondan çıkartma yaparlar. Üreticilerin ve tüketicilerin çalışması karşılıklı dışlama ve senkronizasyon problemlerini içerir.

7.2. Test Uygulaması

JİYİPİZ ile üretici-tüketici problemi ele alınmış ve farklı sayıda üretici ve tüketici ve sabit tampon boyunda elde edilen metrik değerleri tablolarda gösterilmiştir. Testler 1.6 Ghz Pentium Dual Core Intel işlemci ve 2 Gb belleğe ve Ubuntu 10.04 işletim sistemi üzerinde Java 1.6.0_20 çalışma ortamına sahip bir bilgisayar sisteminde gerçekleştirilmiştir. Kullanılan üretici ve tüketici ipliklerin yapısı ve tampon gerçekleştirimi Tablo 3 ve 4'de gösterilmiştir.

Tablo 3: Üretici ve tüketici iplikleri

Üretici	Tüketici

public void run() {
while () {
tampon.put();
}}
	public void run() {
	while () {
	tampon.get();
	}}

Tablo 4: Üretici-tüketici probleminde kullanılan tamponun put ve get metodları

tampon_dolu:tamponun	tampon_boş:tamponun

dolu olduğunu gösterir	boş olduğunu gösterir
Put()	Get()

synchronized void get() {
while (tampon_dolu){
wait();
}
tampon_boyut++;
notifyAll();
if (tampon_boyut sona eşitse) {
tampon_bos = true;
}}}
	synchronized void get() {
	while (tampon_bos) {
	wait();
	}
	tampon_boyut--;
	notifyAll();
	if (tampon_boyut sıfırsa)
	{
	tampon_bos = true;
	}}}

Tablo 5'de 20 birimlik tampon boyunda 2 üretici ve 2 tüketici için metrikler listelenmiştir.

Tablo 5: Tampon boyu 5, 2 Üretici 2 Tüketici ile yapılan test çalışması sonucu JİYİPİZ den elde edilen metrik değerleri

İplik	Cevap Süresi (ms)	Kritik Yol (ms)	Kritik Durum	Kullanım Oranı (%)
Üretici1	29,0	3,8	Bekliyor	0.158
Üretici2	28,6	4,6	Başladı	0.120
Tüketici1	24,2	3,2	Çalışıyor	0.295
Tüketici2	24,0	13,3	Başladı	0.052

Tampon boyu sabit tutulduğunda üretici ve tüketici sayısı arttırılarak elde edilen metrikler Tablo 6'da gösterilmiştir.

Elde edilen değerlere göre ipliklerin cevap süresinde artış olmuştur. Bu durum iplik sayısının ve karşılıklı dışlama nedeniyle beklemelerin artmasıyla açıklanabilir. Buna karşılık kritik yol değerleri de cevap süresine paralel şekilde artmıştır. Bu da üretici ve tüketici iplikler arasında dengenin korunduğun göstermektedir. Kullanım oranları üretici ve tüketicilere göre oranlandığında iplik sayısı arrtığında ipliklerin kullanım oranlarının beklemelerin artması nedeniyle düşmüştür.

Tablo 7'de ise tampon boyu aynı tutularak üretici ipliklerin sayısı azaltılmış ve beklenen şekilde üretici ve ipliklerin çalışma sürelerinin yükseldiği gözlenmiştir. Bunun nedeni de üretici ve tüketici arasındaki dengenin bozulmuş olması ve bu nedenle bekleme sürelerinin artmasıdır. Tüketici ipliklerinin kullanım oranlarının arttığı tespit edilmiştir. Bu da tüketici ipliklerinin sayısının fazla olmasından dolayı çalışıyor durumunda bulunma ve tampon boşsa bekleme durumuna geçme sıklığının daha fazla olduğu şeklinde yorumlanabilir.

Tablolarda gösterilen kritik durumlar kritik yol metriğinin ölçüldüğü durumu göstermektedir. Bu metrik aracılığıyla da en fazla çalışma süresinin hangi

durumda olduğu belirlenebilmektedir. Üretici-tüketici probleminde iplik sayıları eşit arttırıldığında ya da iplikler arasındaki denge bozulduğunda ipliklerin kritik durumlarının genellikler 'Bekliyor' durumunda elde edildiği gözlenmiştir.

Test uygulamasında gösterilen şekilde gerçek uygulamalarda da ipliklerin davranışları JİYİPİZ aracılığıyla elde edilerek iplik sayısı ve ipliklerin uykuda kalma süreleri sistem üzerinde ayarlanabilir. İpliklerin davranışı gerçek sistemde ölçüldüğü için başarımı arttıracak şekilde iplik sayılarının iyileştirilmesi sağlanabilir. JİYİPİZ kullanılarak ipliklerin davranışları hakkında bilgi edinilebilir ve bu sayede JİYİPİZ daha uygun çok iplikli sistemler tasarlanmasına yardımcı olabilir.

Tablo 6: Tampon boyu 5, 4 Üretici 4 Tüketici ile yapılan test çalışması sonucu JİYİPİZ den elde edilen metrik değerleri

İplik	Cevap Süresi (ms)	Kritik Yol (ms)	Kritik Durum	Kullanım Oranı (%)
Üretici1	52,4	7,9	Bekliyor	0.077
Üretici2	52,1	7,8	Bekliyor	0.049
Üretici3	48,1	7,6	Bekliyor	0.063
Üretici4	47,5	4,1	Bekliyor	0.553
Tüketici1	45,6	7,8	Bekliyor	0.111
Tüketici2	44,9	17,6	Başladı	0.040
Tüketici3	27,2	21,4	Başladı	0.015
Tüketici4	6,7	6,6	Başladı	0.005

Tablo 7: Tampon boyu 5, 2 Üretici 4 Tüketici ile yapılan test çalışması sonucu JİYİPİZ den elde edilen metrik değerleri

İplik	Cevap Süresi (ms)	Kritik Yol (ms)	Kritik Durum	Kullanım Oranı (%)
Üretici1	35,3	6,0	Bekliyor	0.135
Üretici2	34,4	6,4	Başladı	0.109
Tüketici1	28,6	3,0	Çalışıyor	0.249
Tüketici2	27,7	12,7	Başladı	0.058
Tüketici3	15,4	10,5	Başladı	0.027
Tüketici4	5,6	5,6	Başladı	0.006

7.3. İplik İzleyici ve Diğer İzleyicilerin Karşılaştırılması

İzleyici gerçekleştirmeleri için en önemli değerlerden bir tanesi de uygulamalara izlerken ne kadar çalışma yükü getirdiğidir. Tablo 8, JİYİPİZ'in bilinen izleyicilerle (Jprofiler[14], Netbeans Profiler[15]) kıyaslamalarını göstermektedir. Üretici-tüketici probleminde 2 üretici ve 2 tüketiciyle yapılan karşılaştırma testlerinin sonuçları milisaniye cinsinden listelenmiştir.

Tablo 8: JİYİPİZ'in bilinen izleyicilerle ek izleme yükü kıyaslaması

İplik	İzleme yok (ms)	JİYİPİZ (ms)	JProfiler (ms)	Netbeans Profiler (ms)
Üretici1	1538.4	2969.4	1830.8	1738.7
Üretici2	1538.4	2969.4	1830.8	1738.7
Tüketici1	1538.5	2969.8	1830.8	1738.7
Tüketici2	1538.5	2969.5	1830.8	1738.7

Bilinen izleyicilerle karşılaştırıldığında JİYİPİZ diğerlerine oranla daha fazla ek izleme yükü getirmesine rağmen her iplik durum değişimini kaydettiği için daha detaylı sonuçlar elde etmektedir.

8. Sonuçlar ve Değerlendirme

Bu çalışmada ilgiye yönelik yaklaşımla iplik izleyici geliştirilmiş ve çok-iplikli sistemlerde sistemden zaman bilgisinin doğru şekilde alınabilmesi için örnek bir çalışma platformunda geliştirilen yöntem uygulanmıştır.

Çalışmada ilgiye yönelik programlama yaklaşımının iplik analizinde nasıl kullanılabileceği gösterilmiş ve diğer bilinen izleyicilerle başarımı kıyaslanmıştır. Ayrıca işlemcilerden en hassas zaman bilgisinin alınabilmesi için JAVA uygulama geliştirme platformudan daha alt seviyeli diller kullanılarak bir yöntem geliştirilmiştir.

JİYİPİZ, işlemciden zaman bilgisi alınabilmesi için kullanılan platforma bağımlı bir alt kütüphaneye ihtiyaç duymaktadır. Bu da yüksek seviye bir uygulama özelliğini kaybetmiştir ancak mevcut yüksek seviye platformlarda istenilen bilgiyi elde edecek bir yapı bulunmamaktadır.

9. Kaynaklar

[1] Waddington, D. G., Roy N., Schmidt D.C., Dynamic Analysis and Profiling of Multi-threaded Systems *Proceedings of the 2nd International Workshop on Social Computing Behavior Modeling*, and Prediction, Phoenix, AZ, Mart 31-Nisan 1, 2009.

[2] Hollingsworth J.K., Miller B.P., Parallel program performance metrics: A comparison and validation. In *Proceedings of Supercomputing*, Kasım 1992.
[3] Waddington, D. G., & Yao, B., High Fidelity C++ Code Transformation. *Proceedings of the 5th Workshop on Language Descriptions, Tools and Applications*, Edinburgh, Scotland, UK, 2005.
[4] Kiczale, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., & Griswold, W. G., An Overview of AspectJ. *Lecture Notes in Computer Science*, 2072, s. 327-355, 2001.
[5] IBM Corporation Whitepaper: Develop Fast, Reliable Code with IBM Rational PurifyPlus, 2003.
[6] Miller, B.P., Callaghan, M.D., Cargille, J.M., Hollingsworth, J.K., Irvin, R.B., & Karavanic, K.L., The Paradyn Parallel Performance Measurement Tool. *IEEE Computer, Cilt. 28*., Baskı 11, s. 37-46, 1995.
[7] Kurtar O., İlgiye Yönelik Yaklaşımla Yazılım Geliştirme,Yüksek Lisans Tezi, İstanbul, 2007.
[8] JBoss AOP – Aspect-Oriented Framework for Java, JBoss AOP Referans Dokümantasyonu, http://www.jboss.org.
[9] AspectWerkz - http://aspectwerkz.codehaus.org/.s
[10] Aspect Oriented Programming with Spring, http://static.springsource.org/spring/docs/2.5.x/reference/aop.html.
[11] AOPBenchmark, http://docs.codehaus.org/display/AW/AOP+Benchmark
[12] Time_Stamp_Counter, http://en.wikipedia.org/wiki/Time_Stamp_Counter
[13] CPUID, http://en.wikipedia.org/wiki/CPUID
[14] Jprofiler –http://www.ej-technologies.com/products/jprofiler/overview.html.
[15] Netbeans Profiler, http://profiler.netbeans.org.