Journal of the Faculty of Engineering and Architecture of Gazi University 37:2 (2022) 771-781

Mühendislik Mimarlık Fakültesi Dergisi
Journal of The Faculty of Engineering and Architecture of Gazi University

Elektronik / Online ISSN : 1304 - 4915
Basılı / Printed ISSN : 1300 - 1884

# Secure cache partitioning in simultaneous multi-threading processors for fairness

Sercan Sarı*, Onur Demir
Department of Computer Engineering, Yeditepe University, Istanbul, 34755, Turkey

**Highlights:**

- Eliminating cache-based side-channel attacks
- Considering fairness metric among threads in SMT processors
- Analyzing performance deterioration of the existing architectures

**Keywords:**

- Secure caches
- Computer architecture
- Cache-based side-channel attacks

**Article Info:**

Research Article
Received: 30.03.2020
Accepted: 20.08.2021

**DOI:**

10.17341/gazimmfd.711396

**Correspondence:**

Author: Sercan Sarı
e-mail: ssari@cse.yeditepe.edu.tr
phone: +90 537 737 7194

**Graphical/Tabular Abstract**

Hardware security gained more attention due to the widespread use of cloud computing and remote execution, where multiple executions share a computer's resources. It is possible to extract confidential information such as cryptographic keys through cache-based side-channel attacks as in Meltdown and Spectre attacks, and as a result, secure cache architectures have become one of the hot research topics in the computer architecture field, today. These architectures come with an inevitable performance penalty since there is always an overhead for hiding information from the attackers. Subsequently, the performance degradation is traded off with the improvement in security. In this study, we analyze cache-based side-channel attacks, and the performance deterioration of the existing architectures and come up with a new solution that improves the fairness of the general framework. We propose a secure cache mechanism that respects fairness among the competing threads within a processor. Figure A represents the FairSDP cache design and the results over 25 workloads on an 8-threaded system.

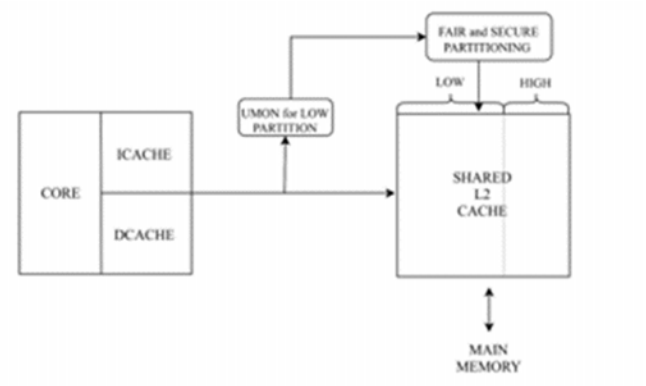



**Figure A.** A. FairSDP Cache Design

**Purpose:** The aim of this study is to design a secure cache mechanism that respects fairness among the competing threads within a processor.

**Theory and Methods:**
We provide a dynamic and secure partitioning mechanism (FairSDP) to defend against cache-based side-channel attacks. We use a utility monitor (UMON) to capture the cache usage behavior of threads to provide a better allocation of partitions on the fly. We evaluate FairSDP architecture in 4-threaded and 8-threaded processors

**Results and Conclusion:**
We show that we can achieve up to 8.7 percent performance improvement over the baseline and 9.2 percent better performance compared to the static partitioning on average, in an 8-threaded system. We also achieve almost identical results in terms of the fairness metric compared to a non-secure dynamic cache partitioning scheme.

We present a fair and secure cache sharing mechanism with dynamic partitioning, FairSDP cache, which provides a fair way of eliminating cache-based side-channel attacks. Although secure cache architectures come with a performance penalty, we demonstrated an improvement to the current secure cache solutions using a number of workloads for 4-threaded and 8-threaded SMT processors in a simulation.

Journal of the Faculty of Engineering and Architecture of Gazi University 37:2 (2022) 771-781

Mühendislik Mimarlık Fakültesi Dergisi
Journal of The Faculty of Engineering and Architecture of Gazi University

Elektronik / Online ISSN : 1304 - 4915
Basılı / Printed ISSN : 1300 - 1884

# Eş zamanlı çok işlem parçacıklı işlemcilerde adalet için güvenli önbellek paylaşımı

Sercan Sarı*, Onur Demir
Yeditepe Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, 34755 Ataşehir, İstanbul, Türkiye

## Ö N E Ç I K A N L A R

- Önbellek tabanlı yan-kanal saldırılarının önlenmesi
- SMT işlemcilerdeki iş parçacıkları arasında adalet metriğinin göz önünde bulundurulması
- Var olan işlemci mimarilerinin performansının analizi ve FairSDP ile karşılaştırılması

Makale Bilgileri
Araştırma Makalesi
Geliş: 30.03.2020
Kabul: 20.08.2021

DOI:
10.17341/gazimmfd.711396

Anahtar Kelimeler:
Bilgisayar mimarisi, güvenli önbellek mimarileri, önbellek tabanlı yan-kanal saldırıları

ÖZ

Bulut bilişimin yaygınlaşması, sanallaştırma ve uzaktan çalıştırma nedeniyle sunucu kaynaklarının birden fazla bilgisayar tarafından paylaşılması, donanımı paylaşılan bir kaynak haline getirmiş ve donanım güvenliği daha fazla önem kazanmaya başlamıştır. Önbellek tabanlı yan-kanal saldırılarıyla şifreleme anahtarları gibi gizli bilgileri çıkarmanın mümkün olduğu hakkında çalışmalar yapılmaktadır. Bu durum güvenli önbellek mimarileri üzerine yapılan çalışmaları arttırmıştır. Ancak güvenli mimariler kaçınılmaz bir performans kaybıyla birlikte gelir, çünkü saldırganlardan bilgi gizlemenin işleme ya da saklama maliyeti vardır. Performansın düşüşünü kullanıcılar arasında adil bir şekilde paylaştırmak, güvenli çözümlerde göz ardı edilmiştir. Bu çalışma ile birlikte, mevcut mimarilerin performans kayıplarını analiz edip, genel çerçevenin adaletliliğini artıran yeni bir çözüm sunuyoruz. Önbellek tabanlı yan-kanal saldırılarına karşı, işlemci içindeki rakip iş parçacıkları arasında adalete saygı duyan güvenli önbellek mekanizması öneriyoruz. Önerilen FairSDP mimarisini 4 ve 8 iş parçacıklı işlemcilerde değerlendirdik. Sonuç olarak, 4 iş parçacıklı bir sistemde ortalama olarak SecDCP mimarisine kıyasla yüzde 38,6 performans artışı elde ediyoruz. 8 iş parçacıklı bir sistemde taban çizgisine göre yüzde 8,7'ye kadar performans iyileştirmesi, statik bölümlemeye kıyasla yüzde 9,2 daha iyi performans ve ortalama olarak SecDCP'ye göre yüzde 14,1 performans artışı sağlayabildiğimizi gösteriyoruz. Ayrıca, güvenli olmayan dinamik önbellek bölümleme tekniğine adalet metriği açısından çok yakın sonuçlar elde ediyoruz.

# Secure cache partitioning in simultaneous multi-threading processors for fairness

## H I G H L I G H T S

- Eliminating cache-based side-channel attacks
- Considering fairness metric among threads in SMT processors
- Analyzing performance deterioration of the existing architectures

Article Info
Research Article
Received: 30.03.2020
Accepted: 20.08.2021

DOI:
10.17341/gazimmfd.711396

Keywords:
Computer architecture, secure cache architectures, cache-based side-channel attacks

ABSTRACT

Hardware security gained more attention due to the widespread use of cloud computing and remote execution, where multiple executions share a computer's resources. It is possible to extract confidential information such as cryptographic keys through cache-based side-channel attacks, and as a result, secure caches have become one of the hot research topics in the computer architecture field today. These architectures come with an inevitable performance penalty since there is always an overhead for hiding information from the attackers. In this study, we analyze cache-based side-channel attacks, and the performance deterioration of the existing architectures and come up with a new solution which improves the fairness of the general framework. We propose a secure cache that respects fairness among the competing threads within a processor. We evaluate our proposed architecture in 4-threaded and 8-threaded processors. We obtain 38.6% performance gain over SecDCP on the average, in a 4-threaded system. In an 8-threaded system, we achieve up to 8.7% performance improvement over the baseline, 9.2% better performance compared to the static partitioning and 14.1% performance gain over SecDCP on the average. We achieve similar results in terms of the fairness metric compared to a non-secure dynamic cache partitioning scheme.

*Sorumlu Yazar/Yazarlar / Corresponding Author/Authors : ssari@cse.yeditepe.edu.tr, odemir@cse.yeditepe.edu.tr / Tel: +90 537 737 7194

## 1. GİRİŞ (INTRODUCTION)

Günümüz dünyasında, bir bilgisayar sisteminin ortak kullanıcıları için en büyük endişelerden biri hassas bilgilerin gizliliğini korumaktır. Bu endişe bulut bilişimin yaygınlaşmasından sonra özellikle artmıştır. Bulut bilişim kullanıcıları, ortak bir donanımı diğer kullanıcılarla paylaşır. Daha önceki dönemlerde kullanıcılar ortak donanımı paylaşmadığında, güvenlik endişesi kriptografik yöntemlerle çözülebilmektedir. Bu yöntemler, veriler ele geçirilse dahi yine de okunamayacağını garanti ediyordu. Ancak, donanım paylaşıldığında, donanım tabanlı saldırıların neden olduğu daha önceden pek düşünülmemiş veri zafiyetleri olabilir. Son zamanlarda ismini çok duyduğumuz donanım tabanlı saldırı tiplerinden biri "yan kanal saldırısı" (side-channel attack) olarak bilinir. Yan kanal saldırıları sistem veya uygulama yazılımındaki zayıflıklardan ziyade, donanımdan edinilen bilgileri kullanır [1]. İşlemlerin ne kadar zaman sürdüğü bilgisi [2], güç kullanımındaki değişimler, elektromanyetik sızıntılar ve hatta donanım tarafından üretilen ses gibi bilgilerinin istismarıyla oluşturulan bir saldırıdır [3]. Bu saldırılar o kadar güçlüdür ki kriptografik anahtarları bile ele geçirebilirler. Örneğin, şifre çözme işlemi sırasında kullandığı güç miktarına bakarak bir akıllı kart üzerinde gizli RSA anahtarı bilgisine ulaşılabilir [4]. Bu noktada, yan kanal saldırılarının özel bir türü olan önbellek tabanlı yan kanal saldırılarının diğer fiziksel yan kanal saldırılarına göre etki alanının daha büyük olduğu iddia edilebilir. Önbelleklerin neredeyse tüm işlemcilerde var olması bu iddia için geçerli bir gerekçedir. Ayrıca, bu saldırılar gerçek donanım üzerinde fiziksel bir yakınlığa ulaşmadan gerçekleştirilebilir. Fiziksel yan kanal saldırılarının donanıma yakın erişime ihtiyaç duyarken, önbellek tabanlı yan kanal saldırıları ise önbellek işlemlerinin ölçülebilir zamanlama bilgisini kullanır [5, 6]. Örneğin, iki işlemin bir L2 önbelleğini paylaştığını varsayalım, işlemlerden saldırgana ait olanı önbellekte bulunan her adresin güncelliğini sorgulayabilir. Diğer işlem önbelleği kullandıktan sonra, saldırgana ait olan işlem önbellekteki adreslerin erişim sürelerini ölçerek herhangi bir adresin geçersiz olup olmadığını anlayabilir. Bu zamanlama bilgisi neredeyse tüm önbelleklerde kolaylıkla ölçülebildiğinden dolayı, bu tür saldırıları önlemek ve ortadan kaldırmak oldukça zordur.

Farklı mimari seviyelerde bu tür saldırıları önlemek için bazı çözümler bulunmaktadır. Son zamanlarda yapılan bazı çalışmalar [7] önbellek yan kanal saldırılarına karşı güvenli önbellek mimarileri önermişlerdir. Hemen hemen tüm çözümler, önbellekte adreslerin geçerliliğini anlamaya yarayan zamanlama bilgilerine ulaşımı iki alternatif yöntemle engellemeye çalışır: ya PLCache [8], Non-Monopolizable (NoMo) önbelleği [9], SecVerilog önbelleği [7], SecDCP önbelleği [10], DAWG [11], ve Catalyst [12] gibi tüm önbelleği bölümlere ayırmak (partitioning) veya RP önbelleği [13], CEASER önbelleği [14], NewCache [15], ScatterCache [16], PhantomCache [17], ve HybCache [18] gibi zamanlama verilerinin rastgele oluşmasını (confuscating) sağlamak. Birinci yöntem verileri mahrem (private) veya herkese açık (public) olarak sınıflandırır ve ardından mahrem bilgilerin önbellekleme konumlarını kısıtlar, böylece zamanlama bilgileri diğer taraflar tarafından görülemez. İkinci yöntem, bellek-önbellek eşlemesini rastgele hale getirmek için şifreleme veya anahtarlama tablosu (hash table) gibi ek önlemler kullanır. Tablo 1'de bazı güvenli önbellekler ve kullandıkları yöntemler özet bir tablo olarak görülebilir.

Paylaşılan önbelleklerde güvenliği sağlamak için verimli ve etkili bir yöntem olan statik önbellek bölümlemede, işlemlerin sadece belirli önbellek bölümlerini kullanmasına izin vererek kısıtlamak işlemci performansını önemli ölçüde azaltır [10]. Genel olarak paylaşılan önbellekte verimliliği artırmak için kullanılan bir yöntem, yarar tabanlı dinamik önbellek bölümleme (UDCP) mekanizmasıdır [19]. Bu yaklaşımla, bölüm (partition) boyutunu işlemlerin önbellek taleplerine göre dinamik olarak değiştirmek mümkündür. Ancak, bu yaklaşım güvenlik için değil performans için tasarlanmıştır ve önbellek yan kanal saldırılarına karşı savunmasızdır. Bu yaklaşımı temel alan SecDCP adlı önceki bir çalışma [10], güvenli bir dinamik önbellek bölümleme mekanizması önermektedir.

Bölümlemeye dayalı SecDCP, PLCache gibi teknikler önbellek tabanlı yan kanal saldırılarını ortadan kaldırmasına rağmen, bu tekniklerin hiçbiri iş parçacıkları arasındaki adil paylaşımı (fairness) dikkate almaz. Adalet ölçüsü, tüm iş parçacıklarının kaynak paylaşımı açısından eşitliğini ölçer [28]. Tüm iş parçacıklarına eşit davranmak, önbellekteki bölümleri ayarlama esnasında ele alınabilecek basit bir sorun değildir. Bölümleme tabanlı çözümler, gizli bilgilerin izole edilmesinde iyi bir iş çıkarırken, adalet metriği tarafından da tespit edilebilen fark edilir bir performans kaybından muzdariptir.

Bulut bilişim ve çoklu iş parçacıklılıkla (multi-threading) ile önbellek paylaşılan bir kaynak haline gelerek öncekinden daha verimli ve güvenli bir şekilde yönetilmesi daha önemli bir hal almaya başlamıştır. Örneğin, IBM'in Power9 işlemcisi [20] 96 iş parçacığı (SMT8) ile 12 çekirdeği veya 96 iş parçacığı (SMT4) ile 24 çekirdeği destekler. Sonuç olarak, adalet ve hizmet kalitesi (QoS) metrikleri de SMT

**Tablo 1.** Güvenli önbellekler ve kullandıkları yöntemler (Secure Cache Architectures)

| Bölümlere ayırma | Rastgele zamanlama |
|---|---|
| PLCache [8]-2008, NoMo [9]-2012 | NewCache [15]-2007, RP önbelleği [13]-2014 |
| SecVerilog [7]-2015, SecDCP [10]-2016 | CEASER [14]-2018, Scatter [16]-2019 |
| Catalyst [12]-2016, DAWG [11]-2018 | PhantomCache [17]-2020, HybCache[18]-2020 |

tabanlı platformlarda performans ve güç metriklerinin yanında çok değerli metrikler haline gelmiştir. Adalet metriğini geliştirmek için bazı çalışmalar vardır [19]. Ancak, daha önce de belirttiğimiz gibi, bu teknikler önbellek yan kanalı saldırılarına karşı hala güvenlik zafiyetleri içermektedir. Literatürde geniş araştırma çalışmaları olmasına rağmen, hiçbiri hem güvenliği hem de adaleti dikkate almamaktadır. Performansa odaklanırken adalet göz ardı edilebilir. Ancak, çoklu iş parçacılılıkta önbelleğin bölünmesi performansı oldukça olumsuz şekilde etkileyecek bir hale gelebilir. Bu sorunla başa çıkmak için yeni bir çözüm sunuyoruz; iş parçacıkları arasında adaleti gözeten güvenilir bir önbellek (FairSDP). Güvenlikle ilgili iş parçacıkları oldukları için mahrem veri ile çalışan iş parçacıklarının genel iş parçacıklarına göre önceliği olması gerektiğini düşünüyoruz. Bölüm boyutunu ayarlarken, yalnızca genel iş parçacığının talebine göre boyutların değiştirilmesinin 2 iş parçacıklı sistemlerde bile haksız durumlara neden olabileceğini gözlemledik. Mahrem veri ile çalışan iş parçacıklarının talebini gözlemlemek, önbellek saldırılarına karşı yeni bir zafiyete neden olduğundan, bu iş parçacıkları için ayrı bir önbellek bölümü ayrılır ve önbelleğin geri kalanı genel iş parçacıkları arasında paylaşılacak şekilde ayarlanır. Böylece adil önbellek paylaşımı ve güvenilir bir dinamik önbellek bölümleme elde edebiliriz.

Önbellek yan kanal saldırılarının engelleyen ve aynı zamanda iş parçacıkları arasında adaleti gözeten FairSDP (Adil ve Güvenilir Dinamik Bölümleme) önbelleği daha önceki bir çalışmamızda tanıtmıştık [21]. Bu çalışmada, performans ve adaleti iki ana metrik olarak ele alıp önbellek yan kanal saldırılarını hedefleyen bölümleme tabanlı önbellek çözümlerinin tasarımına ve değerlendirmesine odaklanıyoruz. Çalışmamızı, literatürde yer alan adaletli ama güvenlik zafiyetleri içeren çalışmalarla ve adaleti gözetmeyen güvenli mimari çalışmalarla kıyaslıyoruz.

Bu çalışmamız çok iş parçacıklı sistemlerin değerlendirilmesi ile [21] ve [30]'da daha önce sunduğumuz çalışmalarımızı genişletmektedir, daha önce sunduğumuz çalışmamızdan temel farklılıklarımız aşağıdaki gibidir:

- Daha önce yaptığımız çalışmada FairSDP isimli çalışmamız sadece UDCP ve statik bölümleme ile kıyaslanmıştır. Dolayısıyla şu anda var olan çok iş parçacıklı güvenli önbellek mekanizmalarıyla gerçek bir kıyaslama imkanımız olmamıştı. Bu yüzden bu çalışmamızın ana katkısı kendi kulvarındaki başka bir çözümle gerçek bir mukayese imkanı sağlamasıdır.
- Bu çalışmada SecDCP çalışması yeniden tasarlanarak gerçekleştirilmiştir. Daha sonra aynı platformda gerçeklenmiş olan FairSDP ve SecDCP iş çıktısı ve adalet metrikleri üzerinden Bölüm 6'da karşılaştırılmıştır.
- SecDCP çalışmasının yeniden tasarlanması, bu çözümün de tıpkı FairSDP gibi, performans odaklı UDCP önbelleği ve güvenlik odaklı statik bölümleme yöntemi ile de kıyaslamaya olanak sağlamıştır.

Her ne kadar önerilen mekanizmamız SecDCP ile benzerlik gösterse de, mevcut durum [9] üzerindeki temel farklarımız aşağıdaki gibidir:

- Yeni mekanizmamızı 4 iş parçacıklı ve 8 iş parçacıklı bir SMT sisteminde değerlendiriyoruz. Diğer çalışmalar yaklaşımlarının ölçeklenebilir olduğunu iddia etseler de, 2'den fazla iş parçacığı olan SMT sistemleri üzerindeki bir değerlendirmede bulunmamışlardır.
- SecDCP dinamik önbellek bölümleme ile önbellek yan kanal saldırılarını ortadan kaldırılsa da, aynı güvenlik düzeyinde olsalar bile önbellek bölümlerini işlemler arasında paylaşmayı kısıtlar. Bu nedenle, her işlem yalnızca kendi önbellek bölgesi ile etkileşime girebilir. FairSDP'de, aynı güvenlik düzeyindeki iş parçacıklarının aralarında önbellek bölgelerini paylaşmasına izin vererek bu kısıtlamayı ortadan kaldırıyoruz. Böylece gizli olmayan bilgilerin daha esnek bir şekilde önbelleği kullanmalarını sağlıyoruz.
- Hem genel iş parçacıkları hem de mahrem iş parçacılarının önbelleğe kapsamlı bir şekilde ihtiyaç duyduğunda ortaya çıkabilecek adaletsiz durumu ortadan kaldırmaya çalışıyoruz. SecDCP sadece genel işlem parçacıklarının ihtiyacını gözettiği için iş parçacıkları arasındaki adaleti garanti etmez. Çünkü mahrem iş parçacıkları önbellek ihtiyacı duymalarına rağmen önbellek ihtiyacı duyan diğer genel iş parçacıkları mahrem iş parçacığının ihtiyacını gözetmeksizin önbelleğin büyük bir bölümünü işgal edebilirler. Örneğin, sistemde önbelleğe ihtiyaç duyan çok sayıda genel işlem olduğunda, SecDCP tüm önbellek yollarını bu genel işlemler arasında dağıtır ve kalan mahrem iş parçacıkları için ayırdığı önbellek yetersiz olabilir. Aksine, FairSDP, adaleti garanti etmek için mahrem iş parçacıklarının her birine sabit sayıda önbellek yolu ayırır ve bu nedenle UDCP tasarımı tarafından ulaşılan adil sonuçları güvenli bir şekilde elde eder.

Bu makale aşağıdaki gibi düzenlenmiştir. Bölüm 2, önbellek tabanlı saldırıların arka planını ve bu saldırılarla ilgili literatürdeki çalışmaları tanıtır. Bölüm 3, ilgili çalışmalar hakkında bilgi vermektedir. 4. Bölüm önerilen FairSDP Önbellek tasarımımızı açıklamaktadır. Bölüm 5, sistemimizi değerlendirmek için kullanılan deneysel metodolojiyi açıklamaktadır. Bölüm 6, deneylerimizin sonuçlarını göstermektedir. Son olarak, Bölüm 7 bu çalışma için varılan sonuçları sunar.

## 2. ÖNCEKİ ÇALIŞMALAR (BACKGROUND INFORMATION)

Bu bölümde, önbellek, önbellek tabanlı yan kanal saldırıları ve bu saldırılara karşı mevcut güvenlik mekanizmaları sunulmaktadır.

### *2.1. Önbellek (Cache)*

Önbellek, bellekten ortalama veri erişim maliyetini azaltmak üzere tasarlanmış bir donanımdır. Önbellek, ana belleğe göre işlemci çekirdeğine göre daha küçük, daha hızlı ve daha yakındır. Sık kullanılan verileri veya işlemci önbelleğindeki en son verileri depolayarak, bellek erişimini hızlandırır. Bellek işlemleri, CPU işlemleriyle karşılaştırıldığında bir darboğaz olduğundan, bellek işlemleri bir hiyerarşi içerisinde yapılır.

Bellek hiyerarşisi, CPU'ya daha yakın olan her düzeyin daha hızlı, daha küçük ve daha pahalı olduğu birden fazla seviyeye ayrılmıştır. Birinci seviye önbellek (L1) performans sorunları nedeniyle doğrudan çekirdeğe bağlanır. Bir von-Neumann mimarisinin komut ve veri için tek bir önbelleği olmasına rağmen, Harvard mimarisi komut ve veri için iki ayrı veri yolu kullanır. İkili önbellekleri olduğundan, bellek işlemlerinin her iki veriyolunda da yapılmasına izin verir ve performansı artırır.

Programlar, önceden kullandıkları adreslere yakın veya eşdeğer veri ve komutları kullanmayı tercih eder. Örneğin, bir program bir döngü kullanıyorsa, aynı kod tekrar tekrar yürütülür. Bu nedenle, belleği daha hızlı hale getirmek için son kullanılan ve sık kullanılan veriler saklanır. Bu, işlemci performansı için önemli bir öneme sahiptir. Ancak, önbelleksiz bir işlemcide bu mümkün olamaz. Önbellek, ana belleğin sadece bir kısmını tutabilir. Bu nedenle, hem verileri hem de verilerin adresini bellekte depolanmalıdır. İşlemcinin belirli bir adresi okuması ya da yazması gerektiğinde önbelleğe bakacaktır. Veriler önbellekte bulunmazsa, bellek hiyerarşisinde L2 önbellek veya ana bellek olabilecek daha düşük bir seviyeden alınır. Performans sorunları nedeniyle, önbellek hattı veya blok aynı anda taşınır çünkü yerellik nedeniyle birlikte ihtiyaç duyulur. Bu, sonraki yükler ve depolamalar için erişim sürelerini azaltır.

### *2.2. Önbellek Tabanlı Yan Kanal Saldırıları* *(Cache-based Side-channel Attacks)*

Önbellek yan kanal saldırıları, saldırgan tarafından ölçülebilen bilgiye göre "erişime bağlı" ve "zamanlamaya bağlı" saldırılar olarak ikiye ayrılır [22]. Erişime bağlı saldırılarda, saldırgan kurbanın önbellek erişimlerinin kendi önbelleğine olan etkisini inceleyerek, kurbanın erişimleri hakkında bilgi edinerek, gizli kalması gereken bilgilere ulaşır. Zamana bağlı saldırılarda, saldırgan, kurban iş parçacığının işlem süresini ölçerek çıkarımlarda bulunur. [23]'te, bu sınıflandırma, saldırıların temel nedenlerini ve potansiyel karşı önlemleri tanımlamak için geliştirilmiştir. Bahsi geçen çalışmada saldırganın bellek içindeki yeri öğrenme yöntemine dayalı yeni bir sınıflandırma daha yapılır: "çekişme(contention) tabanlı saldırılar" ve "tekrar-kullanım tabanlı saldırılar".

[2] ve [24] 'de yazarlar Prime-Probe ve Evict-Time saldırı kavramlarını açıklamıştır. Bu iki saldırının ana noktası, kurban uygulaması tarafından hangi belirli önbellek kümelerine erişildiğini belirlemektir. Aşağıdaki bölümde, önbellek tabanlı yan kanal saldırıları ayrıntılı olarak açıklanmıştır.

#### *2.2.1. Prime probe saldırısı (Prime probe attack) [2]*

Prime-Probe saldırısının ilk adımında, saldırgan bir veya daha fazla belirli önbellek kümesini kendi verileriyle tamamen doldurur. İkinci adımda, saldırgan kurban programını bekler. Bu aşamada kurban çeşitli önbellek kümelerini kullanır ve saldırganın bu lokasyonlardaki verileri geçerliliğini yitirir. Son olarak, saldırgan önbelleğe tekrar eriştiğinde, her veri kümesini yüklemek için gereken zamanı ölçer. Kurbanın değiştirdiği lokasyonlardaki veriler geçerliliğini yitirdiğinden, bu verilerin yükleme süresi daha uzun sürer. Bu şekilde kurban verisinin hangi lokasyonlarda olduğu tespit edilebilir. Algoritma 1 bu tür saldırıların adımlarını özetlemektedir.

*Algoritma 1.* Prime-Probe

1 : Saldırgan belirli önbellek kümelerini kendi verileriyle doldurur.
2 : Saldırgan kurbanın harekete geçmesini bekler. Kurban önbellekteki bazı kümeleri kullanır.
3 : Saldırgan hangi önbellek kümelerinin kurban tarafından erişildiğini belirler.

#### *2.2.2. Evict time saldırısı (Evict time attack) [2]*

Yine [2] 'de açıklanan ikinci saldırı türü Evict-Time saldırısıdır. Kilit nokta, kurbanın eylemi sırasında hangi önbellek kümelerine eriştiğini belirlemektir. Bu nedenle, bu saldırılar [23] taksonomisinde çekişme tabanlı saldırılar olarak sınıflandırılmaktadır. Algoritma 2, bu tür saldırıların adımlarını özetlemektedir. İlk olarak, kurban işleminin süresi saldırgan tarafından ölçülür. İkinci adımda, saldırgan belirli bir önbellek kümesini kendi verileriyle doldurur ve kurbanın verilerini bu önbellek kümesinden çıkarır. Üçüncü adımda saldırgan, kurban programının yürütme süresini tekrar ölçer ve iki ölçüm arasındaki zamanlama farkının olup olmadığını kontrol eder. Bu farkın tespitiyle, kurbanın programı çalışırken belirli bir önbellek kümesinin ne kadar kullanıldığı anlaşılabilir.

*Algoritma 2.* Evict-Time

1 : Saldırgan, kurbanın işlem süresini ölçer.
2 : Saldırgan belirli bir önbellek kümelerini çıkarır.
3 : Saldırgan, kurbanın tekrar işlem süresini ölçer.

#### *2.2.3. Flush reload saldırısı (Flush reload attack) [23]*

Prime-Probe ve Evict-Time saldırılarının aksine, Flush-Reload saldırısı saldırgan ve kurban arasında bir adres alanının paylaşıldığını varsayar. Algoritma 3 bu saldırı tekniğini özetlemektedir. İlk adımda, saldırgan bir önbellek satırını temizler (flush). İkinci adımda, saldırgan kurbanın programını bekler ve üçüncü adımda saldırgan, ilk adımda temizlenen aynı önbellek satırına erişir (reload). Saldırgan işlem süresini ölçerek, erişimin önbellekten mi yoksa ana bellekten mi yüklendiğini anlayabilir. Kurban program Flush-Reload aralığı sırasında güvenlik açısından kritik bazı verilere erişirse, erişim önbellekten yapılacağı için saldırgan çok daha düşük bir yeniden yükleme süresi ölçer.

Gullasch vd. [25] bu saldırı tekniğini AES'in OpenSSL uygulamasına saldırmak için önermiştir. Yarom vd. aynı tekniği kullanarak ilk son seviye önbellek saldırısını uygulamıştır [26].

*Algoritma 3.* Flush-Reload (Temizleme-Yeniden Yükleme)

1 : Saldırgan bir önbellek satırını temizler (önbellekteki güvenlik açısından kritik veriler).
2 : Saldırgan kurbanın harekete geçmesini bekler.
3 : Saldırgan, 1. adımdaki karşılık gelen önbellek satırının kurban tarafından yüklenip yüklenmediğini kontrol eder.

## 3. İLGİLİ ÇALIŞMALAR (RELATED WORK)

Bu bölümde yarar tabanlı önbellek bölümleme [19], statik önbellek bölümleme [7] ve güvenli ve dinamik bölümleme [10] açıklanmaktadır.

### *3.1. Yarar Tabanlı Önbellek Bölümleme (Utility Based Cache Partitioning)*

Yarar Tabanlı Önbellek Bölümleme (Utility Based Cache Partitioning, UDCP) [19], paylaşılan bir önbelleği çalışan iş parçacıklarının taleplerine göre birden çok iş parçacığı arasında bölüştüren bir mekanizmadır. Bu tasarım, önbellek kaynaklarını iş parçacıkları arasında adil bir şekilde paylaşmak için çalışma zamanında her iş parçacığı için yarar monitörleri (UMON) kullanır. UMON her bir iş parçacığı için yarar bilgisi alır ve bölümleme algoritması, önbellekte ayırma yollarının sayısına karar vermek için UMON tarafından toplanan bilgileri kullanır. Bir iş parçacığının yarar bilgisini izlemek için, mümkün olan tüm yolların ıska (miss) sayısı kaydedilmelidir. Bu bilgilere göre, her önbellek için en uygun yol sayısı ayarlanabilir. UDCP, önbelleği rakip iş parçacıkları arasında adil bir şekilde bölmeyi başarsa da, güvenliği hiç dikkate almaz. Bu makalede sunduğumuz çalışma UCDP'yi güvenlik konusunda tamamlamayı hedeflemektedir.

### *3.2. Statik Bölümleme (Static Partitioning)*

Statik önbellek bölümlemesinde [7], kurban ve saldırgan farklı önbellek yollarına sahiptir. Bu yaklaşım temel olarak kurban ve saldırganın önbelleğini bölümlere ayırır. Mahrem ve genel bilgileri ayırmak için Düşük (L) ve Yüksek (H) zamanlama etiketleri kullanılır. Bu etiketler kullanılan veriyi tanımlar. Aynı zamanda, önbellek yollarını güvenlik seviyeleri L ve H arasında bölüştürür. Bütün L bölümü etiketi L olan veriye tahsis edilir. Kalan alan H etiketli veri için kullanılar. Önbellek yan kanal saldırılarını azaltmak için ucuz ve etkili bir yöntem olmasına rağmen, belirli önbellek bölümlerindeki işlemleri kısıtlamak işlemci performansını önemli ölçüde azaltır.

### *3.3. Güvenli ve Dinamik Önbellek Bölümleme (Secure and Dynamic Partitioning)*

Güvenli ve Dinamik Önbellek Bölümleme (SecDCP) statik olarak bölümlemek yerine, önbellek yollarını dinamik olarak bölümler. En az iki güvenlik sınıfı Düşük (L) ve Yüksek (H) desteklenir. SecDCP, L'nin bölüm boyutu artırıldığında veya azaltıldığında azaltılan veya artırılan önbellek hatalarının yüzdesini izleyerek L'ye atanan yolları ayarlar. Önbellek yollarını ayarlarken, L'lerden H'lere bir değişiklik olursa, yeniden kullanmadan önce önbellek satırı temizlenir. Bununla birlikte, H'lerden L'lere bir değişiklik olursa, H çizgileri değiştirilmeden kalır. Bu, L'nin daha önce H'ye atanan önbellek satırlarını okumasına izin vermez.

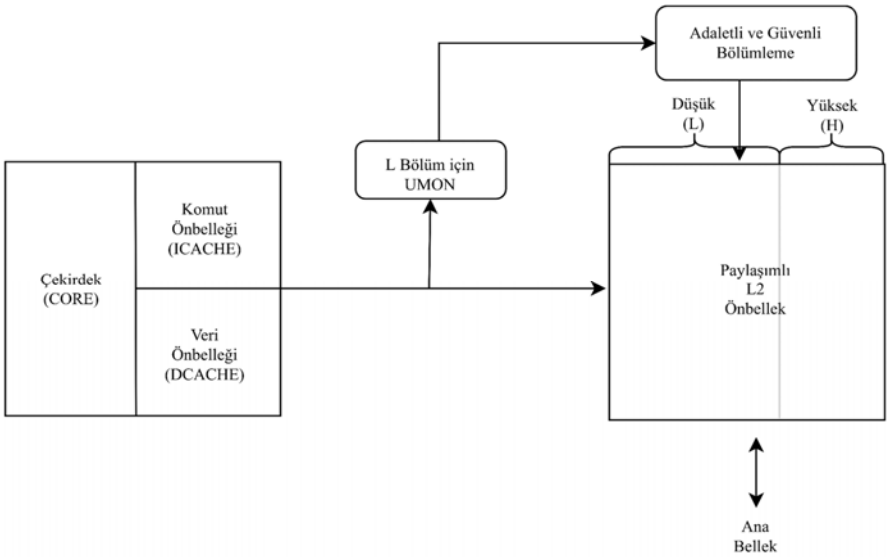
Bu tasarım dinamik önbellek bölümleme ile önbellek yan kanal saldırılarını ortadan kaldırsa da, aynı güvenlik düzeyine sahip olsalar bile önbellek bölümlerini işlemler arasında paylaşmayı kısıtlar. Bu nedenle, her işlem yalnızca kendi önbellek bölgesi ile etkileşime girebilir. Bu, hem genel işlemlerin hem de mahrem işlemlerin önbelleğe kapsamlı bir şekilde ihtiyaç duyması durumunda haksız bir durum oluşturabilir. Mahrem işlem için en az bir önbellek yolu ayrılmasına rağmen, her mahrem işlem yalnızca bir önbellek yolu alabileceğinden, bu adaleti garanti etmez. Örneğin, sistemde önbelleğe ihtiyaç duyulan çok sayıda ortak işlem olduğunda, SecDCP tüm önbellek yollarını bu ortak işlemler arasında dağıtır ve kalan mahrem işlemler için yetersiz önbellek kaynakları bırakır. Bu sorunu Bölüm 4'te açıklanan önerilen tasarım ile gidermeye çalışıyoruz. SecDCP'nin ölçeklenebilir olduğu iddia edilirken, 2'den fazla iş parçacığı olan SMT sistemleri üzerinde bir değerlendirme yapmamışlardır [5]. Bütün çalışmalarda H etiketine sahip iş parçacığı sayısı bir olarak kabul edilmiştir.

Bu üç yöntemi yeniden uyguladık ve önerilen tasarımımızla karşılaştırdık. Zamanlama kanalı saldırılarına karşı güvenli bir önbellek mekanizması sağlarken, aynı anda iş parçacıkları arasında adalet metriğini de göz önünde bulunduran bir çözüm ürettik.

## 4. TASARIM (DESIGN)

FairSDP (Adil ve Güvenli Dinamik Önbellek Bölümleme), dinamik iş parçacığı bölümleme kullanarak önbellek yan kanal saldırılarını ortadan kaldırmayı ve aynı zamanda çok iş parçacıklı bir ortamdaki işlemler arasındaki adaleti göz önünde bulundurmayı amaçlamaktadır. Şekil 1'de gösterildiği gibi, FairSDP paylaşılan önbelleği iki bölgeye ayırır. Gizlilik gerektirmeyen herkese açık verilerin L (DÜŞÜK) ve mahrem verilerin H (YÜKSEK) olarak etiketlendiğini varsayıyoruz. Önbellek, hiçbir L verisi ve H verisi aynı önbellek satırında aynı anda bulunamayacak şekilde bölümlenmiştir. Bu nedenle, bir iş parçacığının zamanlama bilgilerini mahrem bir veriden çıkarımı imkansız olacaktır. Ayrıca, bölümleme eş zamanlı olarak adil bir şekilde yapılarak, böylece hem L hem de H veri bölgeleri yeterli miktarda alana sahip olmasına dikkat edilir.

İşlemler kullandıkları veriye göre yine H ya da L olarak tasnif edilir. Önbellek istatistiklerini toplayabilir ve L işlemlerinin talebini tahmin edebiliriz, ancak güvenlik nedeniyle H işlemleri için aynısını yapmak mümkün değildir. L işlemlerinin önbellekteki anlık taleplerine göre verilecek bölüm alan miktarı ayarlanabilir. Bu arada, bir H işlemi için sabit bir bölüm ayırılır ve L işlemlerinin talebini ölçmek için [19] 'da açıklanan yarar monitörleri (UMON) kullanılır. Her bir iş parçacığı için kullanım istatistikleri, özel UMON işlemi tarafından üretilir. Bu istatistikler, tüm olası önbellek boyutları için önbellek ıskalama (miss) sayısını




**Şekil 1.** FairSDP Tasarımı (Framework of FairSDP Cache)

önbellek boyutunda hesaplamayı sağlar. Ardından, bu bilgileri, kaynakların adil bir şekilde paylaşılması için bölüm alanını ayarlamak için kullanırız.

FairSDP, L ve H işlemlerinin aynı anda bellek yoğun olduğunda ortaya çıkabilecek haksız durumu ortadan kaldırmaya çalışır. Önbellek bölümlendirmeyi yalnızca L işlemlerinin talebini toplayarak ve güvenlik açısından kritik H işlemlerinin önbellek gereksinimlerini göz ardı ederek ayarlamak, güvenli uygulamaların çalıştırılmasında performans sorunlarına neden olabilir. Algoritmamız [19]'da önerilen ileri okuma(look ahead) algoritmasından türetilmiştir. Algoritma 4'te görüldüğü gibi, her H işlemi için bir m-yollu (m-way) statik bölüm ayrılır ve sonra, önbelleğin geri kalanında en uygun bölümlemeyi ayarlamak için kalan L işlemleri için orijinal ileriye okuma(look ahead) algoritmasını uygularız. Burada *N* önbellek ilişkisini(associativity), *Hcount* H işlem sayısını ve *Lcount* L işlem sayısını temsil eder. Çıktı vektörü, *alloc*, her işlem için tahsis edilen maksimum önbellek yolu sayısını saklar. Algoritmanın, orijinal algoritmada olduğu gibi her bir L işlemine en az bir önbellek yolu atayarak başlatılır.

*Algoritma 4.* Önerilen ileri okuma algoritması

```
1  : balance = N - (m * Hcount + 1 * Lcount)
2  : alloc[i] = m for each high process i
3  : alloc[j] = 1 for each low process j
4  : call original lookahead algorithm for all low processes
5  : return alloc
```

Algoritma 5'te gösterildiği gibi güvenli ve adil bölümleme sağlamak için değiştirme politikasında bazı değişiklikler yaptık. Bölümleme yolunu uygulamak için önbelleğe hattı yükleyen çekirdeği tanımlamak için her satırın etiket deposu (tag-store) girişine bir bit ekliyoruz. . Önbellek ıskalamasında , ilk olarak, geçerli önbellek kümesinde ıska bekleyen *i* işleminin önbellek satırlarının sayısını sayarız ve doluluk vektöründe saklarız. İşlem bir L işlemiyse ve bu kümedeki doluluk oranı, işleme ayrılan satır sayısından azsa, işleme ait olmayan tüm satırlar arasında En Son Kullanılan (LRU) satır çıkartılır. Bu sayı maksimum değerine ulaşırsa veya bir H işlemi ise, ıskaya neden olan işlemin tüm satırları arasındaki LRU satırı çıkarılır.

*Algorithm 5.* Önerilen değiştirme algoritması

```
1  :  occupancy[i] = Count(i, set)
2  : if all cache lines are valid then
3  :     if process = L and occupancy[i] < alloc[i] then
4  :             evict LRU line j which belongs to other L
   processes
5  :     else
6  :      evict LRU line j which belongs to current process i
7  : else
8  :     if occupancy[i] < alloc[i] then
9  :         select an invalid line j
10 :     else
11 :      evict LRU line j which belongs to current process i
12 : insert new cache line on line j
```

## 5. DENEYSEL METODOLOJİ (EXPERIMENTAL METHODOLOGY)

FairSDP'yi diğer önbellek mimarileri ile karşılaştırmak ve değerlendirmek için, karşılaştırdığımız tüm sistemleri M-Sim simülatöründe modelledik ve uyguladık [27]. Tablo 2 sistem konfigürasyonunu göstermektedir. Test senaryolarımızı oluşturmak için 8 SPEC CPU2006 deney seti kullandık. Çeşitlilik için *hmmer*, *libquantum*, *mcf*, *milc*, *namd*, *omnettpp*, *sjeng* ve *zeusmp* setleri seçilmiştir. 4 iş parçacıklı karışımlar için 70 iş yükünü ve 8 iş parçacıklı karışımlar için 25 iş yükünü test ettik. Simülasyonları 10 milyon komutu hızlı modda (fast-forward) çalıştırdık ve 250 milyon komut için döngüye-duyarlı (cycle accurate) simülasyonlar gerçekleştirdik.

**Tablo 2.** Sistem Konfigürasyonu (System Configuration)

| | |
|---|---|
| L1 önbellek | Mahrem, 32kB, 2-yollu set associative, split D/I |
| L2 önbellek | Paylaşımlı, 1MB, 8-yollu set associative |
| Bellek | 100-cycle latency |

Önerilen programı Döngü Başına Komut (IPC) ve IPC metriklerinin harmonik ortalamasını kullanarak değerlendirip karşılaştırdık. IPC metriği işlemcinin verimini gösterir, ancak yalnızca IPC'yi kullanmak düşük bir IPC iş parçacığına adil olmayabilir. N iş parçacığının IPC'sini hesaplayan formül Denklem (1) 'de gösterilmiştir. Denklem (2) ve Denklem (3)'te gösterilen IPC'nin harmonik ortalaması, çalışmamızda kullandığımız ikinci metriktir. Bu metrik, adalet metriği olarak da bilinir, çünkü tek bir iş parçacığı haksız muamelenin bir sonucu olarak performans cezası alsa dahi, derhal metriğin değerine yansır.

$$\text{IPC} = \sum \frac{\text{iş parçacığı tarafından işlenen instruction sayısı}}{\text{çevrim sayısı}} \quad (1)$$

$$\text{IPC}'\text{nin harmonik ortalaması} = N / \sum \frac{\text{UDCP}'\text{nin IPC}_i}{\text{SecDCP}'\text{nin IPC}_i} \quad (2)$$

$$\text{IPC}'\text{nin harmonik ortalaması} = N / \sum \frac{\text{UDCP}'\text{nin IPC}_i}{\text{FairSDP}'\text{nin IPC}_i} \quad (3)$$

## 6. SONUÇLAR VE TARTIŞMA
**(RESULTS AND DISCUSSIONS)**

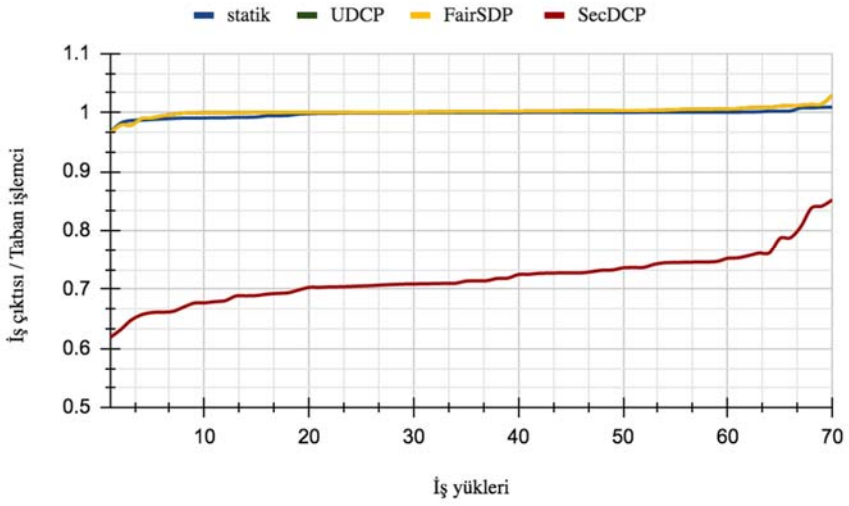
### *6.1. İş çıktısı metriğinde performans*
*(Performance on throughput metric)*

Şekil 2 ve Şekil 3, sırasıyla 4 iş parçacıklı bir sistemde 70 farklı iş yükü ve 8 iş parçacıklı bir sistemde 25 farklı iş yükü arasındaki performans sonuçlarını özetlemektedir. İş çıktısı sonuçlarını LRU değiştirme ilkesini kullanarak 8 yollu set ilişkisel önbellekle bir temel işlemciye normalleştirdik. İş yükleri, temel işlemcideki performanslarına göre sıralanmıştır. FairSDP, UDCP ve statik bölümleme arasında performans açısından önemli bir fark olmamakla birlikte, Şekil 2'de gösterildiği gibi 4 iş parçacıklı bir sistemde SecDCP ile performans arasında önemli bir fark vardır. 4 iş parçacıklı bir sistemde SecDCP'ye göre yüzde 38,6 performans artışı elde ettiğimizi görüyoruz.. 8 iş parçacıklı bir işlemcide, dinamik bölümleme mekanizmaları için bir kazan-kazan durumu ile karşılaştık. Önbellek zaman kanalı saldırılarına karşı korunurken statik bölümlemenin performansını geçiyoruz. Sonuç olarak, taban işlemciye göre yüzde 8,7'ye kadar performans artışı ve statik bölümlemeye kıyasla yüzde 9,2 daha iyi performans ve 8 iş parçacıklı bir sistemde ortalama olarak SecDCP üzerinde yüzde 14,1 performans artışı sağlayabildiğimizi gösterdik.

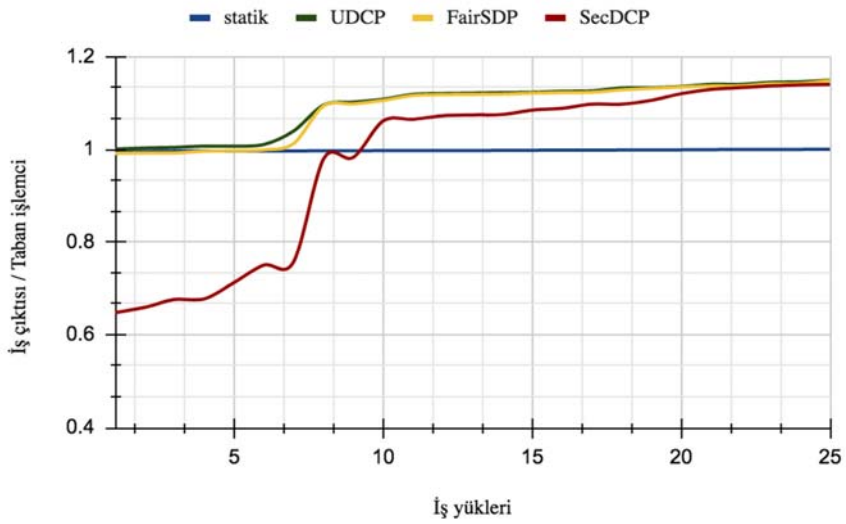
### *6.2. Adalet metriğininin performansı*
*(Performance on fairness metric)*

[28] 'de, normalize edilmiş IPC'lerin harmonik ortalamasının hem adalet hem de performans dikkate alındığı iddia edilmektedir. FairSDP'yi UDCP ile adalet metriğine göre karşılaştırdık [19]. Şekil 4, 8 iş parçacıklı bir sistemde FairSDP'nin UDCP'ye göre performansını göstermektedir. UDCP, statik bölümleme yaklaşımına göre yaklaşık yüzde 10 adalet iyileştirmesi olduğunu iddia ediyor ve sonuçlarımız, güvenli bir önbellek sağlarken, mekanizmamızın UDCP tasarımı kadar adil olduğunu gösteriyor. Ayrıca, iş parçacığı arasındaki adaleti ne kadar geliştirebileceğimizi göstermek için SecDCP'yi adalet ölçüsü açısından UDCP ile karşılaştırıyoruz. Şekil 4, 8 iş parçacıklı bir sistemde UDCP'ye göre SecDCP performansını göstermektedir. Yukarıdaki paragrafta da belirttiğimiz gibi, UDCP statik bölümleme yaklaşımına göre yaklaşık yüzde 10 oranında adalet gelişimi iddia etmektedir. Ancak SecDCP sonuçları, Şekil 4'te görüldüğü gibi FairSDP kadar adil olmadığını göstermektedir.

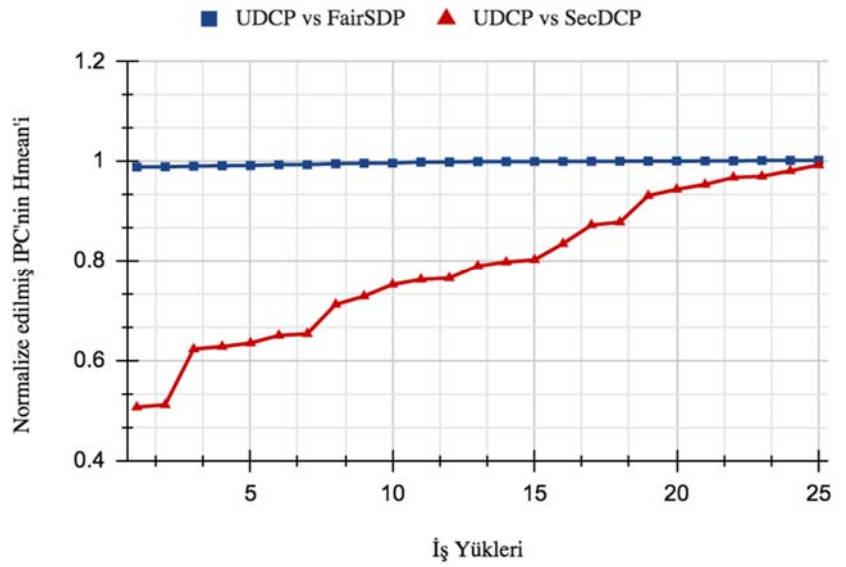
IPC sonuçlarına, örnekleme iş yüklerimizin tüm kriterleri gösterip göstermediğinden emin olmak için bir t-testi [26] uyguladık. T-test iki deney grubu ölçüm sonuçlarının arasında belirgin bir fark olup olmadığını sayısal olarak ifade




**Şekil 2.** 4 iş parçacıklı sistemde iş yüklerinin taban işlemciye göre performansları
(Results over 70 workloads in a 4-threaded system)




**Şekil 3.** 8 iş parçacıklı sistemde iş yüklerinin taban işlemciye göre performansları
(Results over 25 workloads on an 8-threaded system)




**Şekil 4.** İş yüklerine göre normalize edilmiş IPC'nin Hmean'i (FairSDP and SecDCP on fairness metric on an 8-threaded system)

eden bir istatistiki testtir ve gerçekleştirilen bir değişikliğin özgün yöntemle ne kadar farklı olduğunu gösterir. T-testi sonucu oluşan p değerinin düşük olması gözlemlenen değerlerin şans sonucu oluşmadığını doğrular. p değerinin 0,05 değerine eşit ve bu değerden küçük olması sonuçların geçerliliğini ispatlar. T-testi sonuçlarına baktığımızda, FairSDP önbelleğindeki iş yüklerinin IPC sonuçlarının taban çizgisinin ($p < 0,0001$), statik bölümleme ($p < 0,0001$) ve UDCP'den ($p < 0,001$) önemli ölçüde farklı olduğunu gözlemledik. Bu sonuçlar örnekleme iş yüklerimizin tüm sistemi temsil ettiğini desteklemektedir.

## 7. SONUÇLAR (CONCLUSIONS)

Önbellek yan kanal saldırıları, önbellekli tüm işlemciler için ciddi bir güvenlik tehdidi oluşturmaktadır. Bu da gömülü sistemlerden bulut sunucularına kadar tüm bilgisayarlar için güvenlik açığı oluşturmaktadır. Bu çalışma için hedef aldığımız önbellek yan kanal saldırılarında, veriler ana bellek yerine önbellekten yüklendiğinde oluşan erişim zamanı farklılıkları ölçülüp, veri hakkında öngörülerde bulunarak gerçekleştirilebilir. Bu, şifreleme anahtarlarının açığa çıkmasına kadar giden ciddi bilgi sızıntılarına neden olabilir.

Bu çalışmada, bir dinamik bölümleme yöntemi olan adil ve güvenli bir önbellek paylaşım mekanizması olan FairSDP önbelleğini ve değerlendirilmesini sunduk. FairSDP önbellek tabanlı yan kanal saldırılarını ortadan kaldırmada alternatif çalışmalarda göz ardı edilen adil paylaşım temelli bir çözüm sunmaktadır. Bu çalışmada, genel işlemler ve hassas işlemlerin aynı anda bellek yoğun olduğunda ortaya

çıkabilecek önbellek bölünmesindeki haksız paylaşımını ortadan kaldırmaya çalıştık.

Yaptığımız deneyler sonucunda 4 iş parçacıklı bir sistemde ortalama olarak SecDCP'ye göre yüzde 38,6 performans kazancı ve taban çizgisine göre yüzde 8,7'ye kadar performans artışı ve statik bölümlemeye kıyasla yüzde 9,2 daha iyi performans elde ettik. Yine 8 iş parçacıklı bir sistemde ortalama olarak SecDCP'ye göre yüzde 14,1 performans artışı elde ettik. Güvenli olmayan UDCP önbelleği, ortalama olarak, adalet açısından statik bölünmeli önbelleğe göre yaklaşık yüzde 10 iyileşme göstermektedir. Bu neticeler sonucunda FairSDP önbelleğinin performans ve güvenlik beklentilerimizi karşıladığını görmekteyiz.

## TEŞEKKÜR (ACKNOWLEDGEMENT)

Değerli katkılarından ötürü Prof.Dr. Gürhan Küçük'e teşekkür ederiz.

## KAYNAKLAR (REFERENCES)


1. Rebeiro C., Mukhopadhyay D., Bhattacharya S., Timing channels in cryptography: a micro-architectural perspective, Springer, Switzerland, 2014.
2. Osvik D.A., Shamir A., Tromer E., Cache attacks and countermeasures: the case of AES, Cryptographers' track at the RSA conference, Berlin-Germany, 1–20, 2006.
3. Kocher P., Jaffe J., Jun B., Introduction to differential power analysis and related attacks, https://www.rambus.com/security, Erişim tarihi: 30 Mart 2021.
4. Messerges T.S, Dabbish E.A., Sloan R.H., Investigations of power analysis attacks on smartcards, Usenix Workshop on Smartcard Technologies, Chicago-USA, 1999.
5. Lipp M., Schwarz M., Gruss D., Prescher T., Haas W., Fogh A., Hamburg M., Meltdown: Reading kernel memory from user spac, Security Symposium (USENIX), Baltimore-USA, 973-990, 2018.
6. Kocher P., Horn J., Fogh A., Genkin D., Gruss D., Haas W., Yarom Y., Spectre attacks: Exploiting speculative execution, IEEE Symposium on Security and Privacy (SP), San Francisco-USA, 1-19, 2019.
7. Zhang D., Wang Y., Suh G.E., Myers A.C., A hardware design language for timing-sensitive information-flow security, ACM SIGARCH Computer Architecture News, 43 (1), 503–516, 2015.
8. Wang Z., Lee R.B., New cache designs for thwarting software cache-based side channel attacks, ACM SIGARCH Computer Architecture News, 35 (2), 494–505, 2007.
9. Domnitser L., Jaleel A., Loew J., Ponomarev D., Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks, ACM Transactions on Architecture and Code Optimization (TACO), 8 (4), 35, 2012.
10. Wang Y., Ferraiuolo A., Zhang D., Myers A. C., Suh G. E., SecDCP: secure dynamic cache partitioning for efficient timing channel protection, 53nd ACM/EDAC/IEEE Design Automation Conference, Austin-USA, 1-6, 2016.
11. Kiriansky V., Lebedev I., Amarasinghe S., Devadas S., Emer J., Dawg: A defense against cache timing attacks in speculative execution processors, International Symposium on Microarchitecture (MICRO), Fukuoka-Japan, 974–987, 2018.
12. Liu F., Ge Q., Yarom Y., Mckeen F., Rozas C., Heiser G., Lee R.B, Catalyst: Defeating last-level cache side channel attacks in cloud computing, International symposium on high performance computer architecture (HPCA), Barcelona-Spain, 406–418, 2016.
13. Wang Z., Lee R.B., Covert and side channels due to processor architecture, Annual Computer Security Applications Conference (ACSAC), Miami Beach-USA, 473–482, 2006.
14. Qureshi M.K., Ceaser: Mitigating conflict-based cache attacks via encrypted-address and remapping, International Symposium on Microarchitecture (MICRO), Fukuoka-Japan, 775–787, 2018.
15. Wang Z., Lee R.B., A novel cache architecture with enhanced performance and security, International Symposium on Microarchitecture, Lake Como-Italy, 83–93, 2008.
16. Werner M., Unterluggauer T., Giner L., Schwarz M., Gruss D., Mangard S., Scattercache: Thwarting cache attacks via cache set randomization, Security Symposium (USENIX), Santa Clara-USA, 675–692, 2019.
17. Tan Q., Zeng Z., Bu K., Ren K., Phantomcache: Obfuscating cache conflicts with localized randomization, Network and Distributed System Security Symposium (NDSS), San Diego-USA, 2020.
18. Dessouky G., Frassetto T., Sadeghi A.R., Hybcache: Hybrid side-channel-resilient caches for trusted execution environments, Security Symposium (USENIX), Virtual Event, 451–468, 2020.
19. Qureshi M.K., Patt Y.N., Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches, International Symposium on Microarchitecture (MICRO), Orlando-USA, 423–432, 2006.
20. Sadasivam S.K., Thompto B.W., Kalla R., Starke W.J., Ibm power9 processor architecture, IEEE Micro, 37 (2), 40–51, 2017.
21. Sari S., Demir O., Kucuk G., Fairsdp: Fair and secure dynamic cache partitioning, International Conference on Computer Science and Engineering (UBMK), Samsun-Turkey, 469–474, 2019.
22. Page D., Theoretical use of cache memory as a cryptanalytic side-channel, IACR Cryptology ePrint Archive, 2002 (169), 2002.
23. Liu F., Lee R.B., Random fill cache architecture, International Symposium on Microarchitecture (MICRO), Cambridge-UK, 203–215, 2014.

**24.** Percival C., Cache missing for fun and profit, BSDCAN – The BSD Conference, Ottowa-Canada, 2005.
**25.** Gullasch D., Bangerter E., Krenn S., Cache games–bringing access-based cache attacks on aes to practice, Symposium on Security and Privacy, Oakland-USA, 490–505, 2011.
**26.** Yarom T., Falkner K., Flush+ reload: a high resolution, low noise, l3 cache side-channel attack, in Security Symposium (USENIX), Philadelphia-USA, 719–732, 2014.
**27.** Sharkey J., Ponomarev D., Ghose K., M-sim: a flexible, multithreaded architectural simulation environment, Technical report, Department of Computer Science, State University of New York, Binghamton-USA, 2005.
**28.** Luo K, Gummaraju J., Franklin M., Balancing throughput and fairness in smt processors, International Symposium on Performance Analysis of Systems and Software (ISPASS), Singapore, 164–171, 2001.
**29.** William S., The probable error of a mean, Biometrika, 6 (1), 1–25, 1908.
**30.** Sari S., A fair and secure cache architecture for multi-threaded processors, M.Sc. Thesis, Yeditepe University, Graduate School of Natural and Applied Sciences, İstanbul-Turkey, 2019.