TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KISMİ YÜK YENİLEME TEKNİĞİ İLE SATIRDARBESİ ÖNLEME MEKANİZMALARININ PERFORMANSININ İYİLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Yahya Can TUĞRUL

Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Oğuz ERGİN

AĞUSTOS 2024

ÖZET

Yüksek Lisans Tezi

KISMİ YÜK YENİLEME TEKNİĞİ İLE SATIRDARBESİ ÖNLEME MEKANİZMALARININ PERFORMANSININ İYİLEŞTİRİLMESİ

Yahya Can TUĞRUL

TOBB Ekonomi ve Teknoloji Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Oğuz ERGİN

Tarih: AĞUSTOS 2024

Modern DRAM çiplerinde gerçekleşen okuma hataları, bellek yalıtımını bozabilen yaygın bir zayıflıktır. Bellek yalıtımı, bellek güvenliği ve güvenilirliğinin temel yapı taşlarından biridir. SatırDarbesi zafiyeti, DRAM'deki okuma hatalarına önemli bir örnektir; burada DRAM hücrelerinin bir satırına (DRAM satırı) tekrar tekrar erişmek (darbeleme), fiziksel olarak yakın DRAM satırlarında (kurban satırlar) bit hatalarına neden olur. Ne yazık ki, teknoloji düğüm boyutunun küçülmesi SatırDarbesi zafiyetini kötüleştirir ve böylece, daha yeni DRAM çip nesillerinde daha az erişim yaparak SatırDarbesi bit hataları oluşturulabilir. Güvenilir DRAM kullanımı için, son teknoloji SatırDarbesi önleme mekanizmaları potansiyel kurban satırlardaki DRAM hücrelerinin yükünü yeniler (önleyici yenileme ya da yük yenileme). Gerçekleştirilen bu önleyici yenileme operasyonu DRAM çipini bir süre kilitleyerek performans kayıplarına yol açar. Daha yeni DRAM çip nesilleri ile bu mekanizmalar önleyici yenilemeyi daha sık gerçekleştirir ve daha büyük performans kayıplarına neden olur. Güvenilirlik ve güvenlikten ödün vermeden bu kaybı azaltmak için bizim amacımız, önleyici yenileme için harcanan zamanı azaltmak ve bu azaltılmış zamanın SatırDarbe zafiyeti üzerindeki etkisini anlamaktır. Bu amaçla, gerçek DRAM çiplerinde veri tutma süresi, yük yenileme ve SatırDarbesi zafiyetinin özellikleri arasındaki etkileşimlere dair ilk kapsamlı deneysel çalışmayı sunuyoruz. Üç büyük DRAM üreticisinden toplam 388 DDR4 DRAM çipini test ediyoruz ve bir yük yenileme operasyonunun gecikmesinin önemli ölçüde azaltılabileceğini (%64 oranında) ve dolayısıyla, biraz daha fazla (%0,54 oranında) önleyici yenileme operasyonu gerektireceğini

gözlemliyoruz. Bu gözlemimizi kullanarak, yük yenileme gecikmesini dinamik olarak ayarlayan, mevcut SatırDarbesi önleme mekanizmalarının saldırganlığını düzenleyen ve bellek kontrolcüsü tabanlı düşük maliyetli bir mekanizma olan Kısmi Yük Yenileme ile Agresif Önleme'yi (PaCRAM) öneriyoruz. PaCRAM'i beş son teknoloji SatırDarbesi önleme mekanizması ile kullanarak önleme mekanizmalarının performans (enerji) kayıplarını sırasıyla ortalama olarak %18,95 (%14,59), %12,28 (%11,56), %2,07 (%1,15), %2,56 (%2,18) ve %5,37 (%4,50) oranında azaltarak sistem performansını ve enerji verimliliğini önemli ölçüde artırıyoruz.

Keywords: Bellek, DRAM, SatırDarbesi, Veri tutma hataları, Yük yenileme, Önleyici yenileme, DRAM karakterizasyonu.

ABSTRACT

Master of Science

PARTIAL CHARGE RESTORATION TECHNIQUE TO IMPROVE THE PERFORMANCE OF ROWHAMMER MITIGATION MECHANISMS

Yahya Can TUĞRUL

TOBB University of Economics and Technology
Institute of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Prof. Dr. Oğuz ERGİN

Date: AUGUST 2024

Read disturbance in modern DRAM chips is a widespread weakness that is used for breaking memory isolation, one of the fundamental building blocks of security and privacy in memory. RowHammer is a prime example of read disturbance in DRAM where repeatedly accessing (hammering) a row of DRAM cells (DRAM row) induces bitflips in physically nearby DRAM rows (victim rows). Unfortunately, shrinking technology node size exacerbates RowHammer and as such, significantly fewer accesses can induce bitflips with newer DRAM chip generations. To ensure reliable DRAM operation, state-of-the-art mitigation mechanisms restore the charge in potential victim rows (i.e., preventive refresh or charge restoration). With newer DRAM chip generations, these mechanisms perform preventive refresh more aggressively and cause larger performance overheads.

To reduce this overhead without sacrificing reliability, security, and safety, our goal is to reduce time spent for preventive refreshes and understand this reduced time's impact on RowHammer. To this end, we present the first rigorous experimental study on the interactions between data retention time, refresh, and RowHammer characteristics in real DRAM chips. We test 388 DDR4 DRAM chips from three major manufacturers and observe that a preventive refresh operation's latency can be significantly reduced (by 64%) at the expense of requiring slightly more (by 0.54%) preventive refresh operations. To leverage this observation, we propose Partial Charge Restoration for Aggressive Mitigation (PaCRAM), a memory controller-based low-cost mechanism that dynamically tunes the refresh latency and adjusts the aggressiveness of existing

RowHammer solutions. PaCRAM significantly improves system performance (energy efficiency) by reducing the overhead induced by five RowHammer solutions by 18.95% (14.59%), 12.28% (11.56%), 2.07% (1.15%), 2.56% (2.18%), and 5.37% (4.50%), on average.

Keywords: Memory, DRAM, RowHammer, Data retention failures, Charge restoration, Preventive refresh, DRAM characterization.

İÇİNDEKİLER

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

KISALTMALAR

AH	:	Ağırlıklı Hızlanma (-*ing*, Weighted Speedup)
BER	:	Bit Hata Oranı (-*ing*, Bit Error Rate)
CPU	:	İşlemci (-*ing*, Central Processing Unit)
ÇBB	:	Çevrim Başına Buyruk (-*ing*, Instruction Per Cycle)
DDR	:	Çifte Veri Hızı (-*ing*, Double Data Rate)
DRAM	:	Dinamik Rastgele Erişimli Bellek (-*ing*, Dynamic Random Access Memory)
ECC	:	Hata Düzeltme Kodları (-*ing*, Error Correction Codes)
FPGA	:	Alanda Programlanabilir Kapı Dizileri (-*ing*, Field Programmable Gate Arrays)
FR	:	Tamamen Yenilenmiş (-*ing*, Fully Restored)
FR-FCFS	:	İlk Hazır - İlk Gelen İlk Hizmet (-*ing*, First Ready - First Come First Serve)
GND	:	Toprak Gerilimi (-*ing*, Ground Voltage)
JEDEC	:	Joint Electron Device Engineering Council
$\mathbf{N_{PCR}}$	:	Kısmi Yük Yenileme Sayısı (-*ing*, Number of Partial Charge Restorations)
$\mathbf{N_{RH}}$	:	SatırDarbesi Eşiği (-*ing*, RowHammer Threshold)
PaCRAM	:	Kısmi Yük Yenileme ile Agresif Önleme (-*ing*, Partial Charge Restoration for Aggressive Mitigation)
PCIe	:	Peripheral Component Interconnect Express
PRAC	:	Satır Başına Etkinleştirme Sayma (-*ing*, Per Row Activation Counting)
RFM	:	Yenileme Yöntemi (-*ing*, Refresh Management)
SPD	:	Seri Modül Tanıma (-*ing*, Serial Presence Detect)
SRAM	:	Statik Rastgele Erişimli Bellek (-*ing*, Static Random Access Memory)
$\mathbf{t_{FR}}$	:	FR Yenileme Periyodu (-*ing*, FR Refreshing Period)
$\mathbf{t_{RAS}}$	:	Satır Adresi Gecikmesi (-*ing*, Row Address Strobe)
$\mathbf{t_{RCD}}$	:	Satır Sütun Gecikmesi (-*ing*, RAS to CAS Delay)
$\mathbf{t_{REFI}}$	:	Yenileme Aralığı (-*ing*, Refresh Interval)
$\mathbf{t_{REFW}}$	:	Yenileme Penceresi (-*ing*, Refresh Window)
$\mathbf{t_{RFC}}$	:	Yenileme Gecikmesi (-*ing*, Refresh Cycle Time)
$\mathbf{t_{RP}}$	:	Satır Kapama Gecikmesi (-*ing*, Row Precharge Time)
$\mathbf{th_{PCR}}$	:	Maksimum Kısmi Yük Yenileme Limiti (-*ing*, Partial Charge Restoration Threshold)
TRR	:	Hedefli Satır Yenileme (-*ing*, Target Row Refresh)
VDD	:	Besleme Gerilimi (-*ing*, Supply Voltage)
WCDP	:	En Etkili Veri Deseni (-*ing*, Worst Case Data Pattern)

SEMBOL LİSTESİ

Bu çalışmada kullanılmış olan simgeler açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler	**Açıklama**
mm^2	milimetre kare
ns	nanosaniye
μs	mikrosaniye
ms	milisaniye

1. GİRİŞ

Sistem güvenilirliğini ve güvenliğini sağlamak için bellek yalıtımını (-*ing*, memory isolation) sağlamak kritik öneme sahiptir. Bellek izolasyonu, bir bellek adresine yapılan erişimin, diğer adreslerde depolanan verilere istenmeyen yan etkiler oluşturmamasıdır. Ne yazık ki, agresif teknoloji düğümü ilerlemesi ile birlikte, yaygın olarak kullanılan ana bellek teknolojisi olan dinamik rastgele erişimli bellek (DRAM) [41], artan okuma hatalarından (-*ing*, read disturbance) muzdariptir. Bu durum, bir DRAM hücresine (-*ing*, DRAM cell) erişmenin (okumanın), fiziksel olarak yakın ve erişilmemiş diğer DRAM hücrelerinin veri bütünlüğünü bozması anlamına gelir. Bu bozulmalar, iki ana nedenden kaynaklanır: 1) DRAM hücreleri doğaları gereği yük sızdırır ve zamanla veri kaybeder (veri tutma süresi (-*ing*, data retention time)) ve 2) bir DRAM hücresini okumak, fiziksel olarak yakın olan DRAM hücrelerindeki doğuştan gelen yük sızdırmasını daha da kötüleştirerek, bu hücrelerin veri tutma sürelerinde büyük bir azalmaya yol açar. SatırDarbesi (-*ing*, RowHammer), bu tür DRAM okuma hatalarının çarpıcı bir örneğidir. SatırDarbesi saldırılarında, bir DRAM hücresi satırı (DRAM satırı (-*ing*, DRAM row)), fiziksel olarak yakın bir başka DRAM satırının (saldırgan satır (-*ing*, aggressor row)) tekrar tekrar açılması (darbeleme (-*ing*, hammering)) durumunda bit hataları yaşayabilir [1, 2, 16, 18, 19, 21, 23, 24, 28, 36–38, 46, 48, 51, 52, 57, 61, 62, 70, 75, 81, 82, 92, 101, 110, 114, 116, 119, 128, 129, 131, 133, 146, 148, 149, 153, 159, 161, 164, 165, 169–171, 175, 176, 178, 181, 182, 191, 192, 204–207, 210, 211, 213, 216, 221, 224, 230, 231, 233, 236, 238, 240, 241].

Birçok çalışma, okuma hatalarını kullanarak ayrıcalıkları yükseltmek, özel verileri sızdırmak ve kritik uygulama çıktılarıyla oynamak için geniş bir yelpazede sistemlerde güvenilir saldırılar gerçekleştirdiklerini göstermektedir [1, 8, 16, 18, 19, 21, 23–25, 28, 36–38, 45, 46, 48, 51, 52, 61, 62, 70, 75, 79, 81, 82, 92, 99, 110, 114, 116, 119, 125, 127, 131, 133, 140, 145, 146, 148, 159, 161, 164, 165, 169–171, 173, 175–178, 180, 182, 188, 191, 192, 201, 204–207, 210, 211, 213, 214, 216, 221, 229–231, 236, 238, 240, 241]. Bu saldırılar, sistem güvenliği ve veri bütünlüğü açısından ciddi tehditler oluşturur çünkü saldırganlar bellek yalıtımını ihlal ederek yetkisiz erişim ve veri manipülasyonu yapabilirler. Örneğin, bir saldırgan okuma hatalarını kullanarak bir sistemdeki kritik verilere erişebilir ve bu verileri değiştirebilir veya silebilir, bu da sistemin güvenilirliğini ve işlevselliğini tehlikeye atar. Bu tür saldırılar, özellikle finansal sistemler, sağlık hizmetleri ve devlet kurumları gibi yüksek güvenlik gerektiren ortamlarda büyük risk taşır ve bilgi sızdırma, yetkisiz işlem gerçekleştirme gibi çeşitli zararlı faaliyetlere yol açabilir. Ayrıca, saldırganlar okuma hataları aracılığıyla sistemdeki zayıflıkları kullanarak daha yüksek ayrıcalıklara sahip olabilir ve böylece daha geniş bir alanda hasar verebilirler. Daha da kötüsü, son deneysel çalışmalar [37, 52, 110, 114, 136, 146, 148], daha yeni DRAM çip (-*ing*, DRAM chip) nesillerinin okuma hatalarına karşı daha hassas olduğunu ortaya koymuştur. Bu hassasiyet, teknoloji düğüm boyutlarının küçülmesi ve DRAM hücrelerinin daha yoğun paketlenmesi ile ilişkilidir. Bu durum, saldırganların daha az

çaba ile daha büyük hasar verebilmesini mümkün kılar. Örneğin, 2018-2020 yılları arasında üretilen DRAM çipleri, 2012-2013 yılları arasında üretilen çiplere kıyasla önemli oranda daha az satır aktivasyonundan sonra SatırDarbesi bit hataları yaşayabilir [110]. Bu durum, daha yeni çiplerin daha az dayanıklı olduğunu ve daha kolay saldırıya açık hale geldiğini göstermektedir. Bu bağlamda, daha yeni nesil DRAM çiplerinde okuma hatalarının etkilerini azaltmak ve sistem güvenliğini sağlamak için daha güçlü ve etkili önlemler geliştirilmesi gerektiği açıktır. Yenilikçi çözümler geliştirilmediği takdirde, saldırganlar bu zayıflıkları kullanarak kritik sistemlerde ciddi hasarlara yol açabilir ve bu da kullanıcıların güvenliğini ve gizliliğini tehlikeye atar.

Güvenilir, güvenli ve emniyetli bir DRAM işlemi sağlamak için, geçmiş çalışmalar [5, 7, 12–14, 42, 52, 70, 94, 97, 106, 112, 114, 123, 124, 140, 166, 174, 193, 194, 198, 223, 232, 239], bir saldırgan satır, SatırDarbesi eşik değeri (-*ing*, RowHammer threshold) olarak adlandırılan bir değerinden daha fazla etkinleştirilmeden önce, kurban satırlardaki (-*ing*, victim rows) yükü yenileyen önleyici yenileme (-*ing*, preventive refresh) adı verilen bir işlem gerçekleştirir. Bir DRAM hücresinin yük yenilemesi (-*ing*, charge restoration), ihmal edilemeyecek bir gecikmeye sahiptir ve bu gecikme genellikle $35 - 40ns$ arasında değişir [84–90]. Bir DRAM satırındaki hücrelerin yük yenilemesi sırasında, satırı içeren DRAM bankası (-*ing*, DRAM bank) kullanılamaz hale gelir ve bu nedenle bir DRAM satırının yenilenmesi bellek erişimlerini geciktirerek performans kayıplarına neden olur. DRAM çiplerindeki okuma hataları kötüleştikçe, SatırDarbesi önleme mekanizmaları önleyici yenileme işlemlerini daha agresif (yani, daha sık) bir şekilde gerçekleştirir ve daha büyük performans kayıplarına yol açar [110, 166, 226]. Düşük performans kayıpları ile okuma hatalarını hafifletmek için, yük yenileme işlemini ve bu işlemin DRAM çiplerindeki okuma hatası üzerindeki etkisini daha derinlemesine anlayarak, mevcut ve gelecekteki DRAM tabanlı bellek sistemlerinin güvenilirliği, güvenliği ve emniyeti için daha etkili ve verimli çözümler geliştirmek kritik öneme sahiptir. Geçmiş çalışmalar [2, 9, 36, 52, 56, 57, 72, 93, 101, 110, 114, 128–130, 136, 146–149, 153, 154, 156, 159, 161, 163–165, 177, 181, 213, 224, 227, 228, 230, 233, 242], DRAM okuma hatalarının çeşitli yönlerini incelemiştir. Ancak, veri tutma süresi, yük yenileme ve okuma hataları DRAM hücrelerindeki doğuştan gelen yük sızdırmasıyla sıkı bir şekilde bağlantılı olmasına rağmen, bu etkileşimi ve bu etkileşimin SatırDarbesi savunmaları üzerindeki etkilerini detaylı bir şekilde inceleyen hiçbir çalışma bulunmamaktadır.

Bu tezdeki amacımız, 1) veri tutma süresi, yük yenileme ve okuma hatası özellikleri arasındaki etkileşimleri anlamak ve 2) önleyici yük yenileme işlemleri için harcanan zamanı ve dolayısıyla performans kaybını güvenli bir şekilde azaltmaktır. Bu çalışmada, 1) gerçek modern DRAM çiplerinin veri tutma süresi, yük yenileme ve okuma hatası özellikleri arasındaki etkileşimlere dair ilk kapsamlı deneysel çalışmayı sunuyoruz ve 2) mevcut SatırDarbesi önleme mekanizmalarının performans kaybını, güvenlik garantilerini tehlikeye atmadan azaltmak için önleyici yük yenileme işlemlerinin gecikmesini dinamik olarak ayarlayan yeni bir mekanizma olan Kısmi Yük Yenileme ile Agresif Önleme'yi (PaCRAM) öneriyoruz. Ayrıca, PaCRAM mekanizmasının nasıl çalıştığını, mevcut SatırDarbesi önleme mekanizmalarıyla nasıl entegre edilebileceğini ve bu mekanizmanın performans kayıplarını nasıl etkili

bir şekilde azaltabileceğini ayrıntılı olarak açıklıyoruz. Sonuç olarak, önerdiğimiz yaklaşımın, DRAM tabanlı bellek sistemlerinin güvenilirliği, güvenliği ve performansı açısından önemli iyileştirmeler sağlayabileceğini göstermeyi amaçlıyoruz.

Karakterizasyon. Modern DRAM çiplerinin veri tutma süresi, yük yenileme ve SatırDarbesi zafiyeti arasındaki etkileşimleri anlamak için 388 DDR4 DRAM çipi üzerinde gerçekleştirilmiş ilk kapsamlı deneysel karakterizasyon çalışmasını sunuyoruz. Bu çalışmayı dört set deney yaparak gerçekleştiriyoruz ve bu deneyler sonucunda şu dört çıkarıma ulaşıyoruz. Birincisi, bir DRAM satırına darbeleme yapmak, satırın veri tutma süresini azaltır ve bu nedenle darbelemeyle oluşan okuma hatası, tek başına bir bit hatasına neden olmasa bile, artan yük sızdırması nedeniyle nihayetinde bit hatalarına yol açabilir. Bu nedenle, önceki çalışmaların önerdiği yöntemlerin aksine [110, 114, 161, 224], SatırDarbesi zafiyeti, darbeleme satırlarının neden olduğu azalan veri tutma süresi göz önünde bulundurularak ölçülmelidir çünkü bu çalışmalar kurban satırı, SatırDarbesi testinden hemen sonra veri tutma süresinin etkisini dikkate almadan okumaktadır. İkincisi, kurban satırının yük yenileme gecikmesini azaltmak, hücre yükünün kısmen yenilenmesine neden olabilir ve bu da ya 1) satırın SatırDarbesi zafiyetinde değişikliğe yol açmaz ya da 2) SatırDarbesi eşik değerinin azalmasına neden olur. Üçüncüsü, kısmi yük yenilemenin tekrarlanması, birkaç tekrardan sonra doygunluğa ulaşan bir SatırDarbesi eşiği azalmasına neden olur. Bu nedenle, kısmi yük yenileme, SatırDarbesi zafiyetini daha da kötüleştirmeden binlerce kez gerçekleştirilebilir. Dördüncüsü, yük yenileme gecikmesini belirli bir eşiğe kadar azaltmak, veri tutma hataları üzerinde hiçbir etkiye sahip değildir, ancak gecikmeyi bu eşikten daha fazla azaltmak, veri tutma hatalarını önemli ölçüde kötüleştirir.

SatırDarbesi önleyici yenileme gecikmesini azaltma. Mevcut SatırDarbesi önleme mekanizmalarının neden olduğu önleyici yenileme işlemlerinin gecikmesini dinamik olarak ayarlayan yeni bir mekanizma olan Kısmi Yük Yenileme ile Agresif Önleme'yi (PaCRAM) öneriyoruz. PaCRAM, DRAM çipini değiştirmeden, işlemci çipindeki bellek kontrolcüsü içinde mevcut SatırDarbesi önleme mekanizmaları ile birlikte çalışır. PaCRAM, önleyici yük yenilemelerin gecikmesini dikkatlice azaltır ve SatırDarbesi savunmalarının saldırganlığını ayarlayarak güvenli bir DRAM işlemi sağlar. PaCRAM'ı, iki gerçek DRAM modülünün gerçek DRAM karakterizasyonu gözlemlerimize dayanarak konfigüre ederek, güvenilir ve emniyetli bir DRAM işlemi garanti ediyoruz. PaCRAM'ın, beş son teknoloji SatırDarbesi savunması olan PARA [114], RFM [89], PRAC [91], Hydra [174] ve Graphene [166] 'nin neden olduğu performans (enerji) kaybını 62 test programında ortalama olarak sırasıyla %18,95 (%14,59), %12,28 (%11,56), %2,07 (%1,15), %2,56 (%2,18) ve %5,37 (%4,50) oranında azaltarak sistem performansını ve enerji verimliliğini önemli ölçüde artırdığını gösteriyoruz.

Tez çalışmamızda aşağıdaki katkıları yapıyoruz:

- Veri tutma süresi, yük yenileme ve SatırDarbesi zafiyeti arasındaki etkileşimlerin ilk kapsamlı karakterizasyonunu sunuyoruz. 388 gerçek DDR4 DRAM çipi üzerindeki deneysel sonuçlarımız, yük yenileme işleminin gecikmesini güvenli

bir şekilde azaltabileceğimizi gösteriyoruz.

- Mevcut SatırDarbesi önleme mekanizmalarının sıkça gerçekleştirdiği yük önleyici yenileme işlemlerinin gecikmesini dinamik olarak ayarlayan yeni bir mekanizma olan Kısmi Yük Yenileme ile Agresif Önleme'yi (PaCRAM) öneriyoruz.
- PaCRAM'ın faydalarını beş farklı son teknoloji SatırDarbesi önleme mekanizması kullanarak sergiliyoruz ve PaCRAM'ın önleme mekanizmalarının neden olduğu performans kayıplarını önemli ölçüde azalttığını gösteriyoruz.

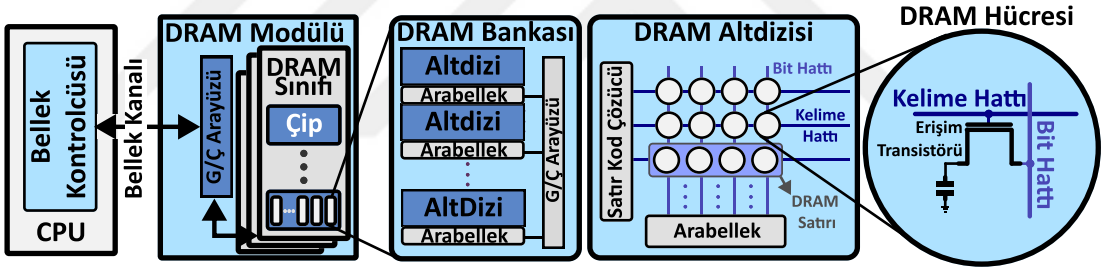
Bu tezin geri kalanı aşağıdaki şekilde organize edilmiştir. Bölüm 2, DRAM organizasyonu, işleyişi ve okuma hataları ile ilgili gerekli arka plan bilgisini verir. Bölüm 3, son teknoloji SatırDarbesi önleme mekanizmalarının motivasyonel bir analizini sunar. Bölüm 4, karakterizasyon metodolojimizi ve test algoritmamızı açıklar. Bölüm 5, veri tutma hataları, yük yenileme ve SatırDarbesi zafiyetinin etkileşimlerinin ilk kapsamlı karakterizasyonunu sunar. Bölüm 6, mevcut önleyici yük yenileme tabanlı SatırDarbesi önleme mekanizmalarının performans kaybını kısmi yük yenileme tekniğinden yararlanarak azaltmak için yeni bir bellek kontrolcüsü tabanlı mekanizma olan Kısmi Yük Yenileme ile Agresif Önleme'yi (PaCRAM) açıklar. Bölüm 7, PaCRAM'ın sistem performansı ve enerji tüketimi üzerindeki faydalarını gösterir. Bölüm 8, azaltılmış gecikme altında DRAM işlemleri, SatırDarbesi zafiyetinin karakterizasyonu ve SatırDarbesi saldırıları ve savunmaları üzerine ilgili geçmiş çalışmaları tartışır. Son olarak, Bölüm 9, tez çalışmasını sonuçlandırır.

2. TEMEL BİLGİLER

Bu bölümde okuyucunun karakterizasyonumuzu ve PaCRAM'i anlaması için gerekli olan DRAM organizasyonu, operasyonları ve okuma hataları hakkında temel bilgileri veriyoruz.

2.1 DRAM Organizasyonu ve İşlemleri

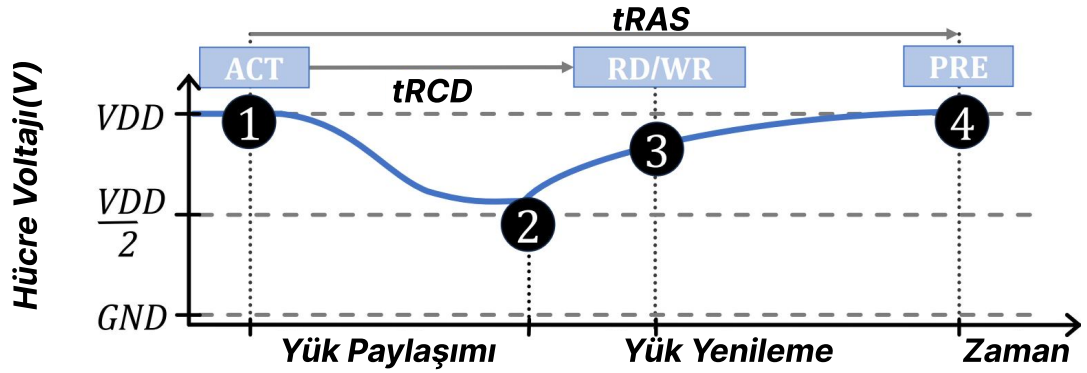
DRAM organizasyonu. Şekil 2.1, modern bir DRAM sisteminin tipik organizasyonunu göstermektedir. DRAM, her biri bir bit veri tutan milyarlarca DRAM hücresinin hiyerarşik bir dizisi halinde organize edilmiştir. Modern sistemlerde, bir CPU çipi bir bellek kontrolcü (*-ing*, memory controller) kullanarak, bellek kanalları (*-ing*, memory channel) aracığıyla bir ya da daha fazla DRAM modülüne (*-ing*, memory module) erişebilir. Her modül, bir veya daha fazla DRAM sınıfından (*-ing*, DRAM rank) oluşur. Bir sınıf, eşzamanlı bir şekilde çalışan ve kanalın giriş/çıkış (G/Ç) arayüzüne (*-ing*, I/O interface) bağlı olan birden fazla DRAM çipinden oluşur.



Şekil 2.1: DRAM organizasyonu.

Bir DRAM çipi, dahili bir veri/komut yolunu (*-ing*, data/command bus) paylaşan birden fazla DRAM bankasından oluşur. Bir DRAM bankası içinde, DRAM hücreleri, her biri yoğun iki boyutlu DRAM hücrelerinden oluşan çok sayıda altdiziye (*-ing*, subarray) (örneğin, 128) ve altdizi içindeki verileri işlemek için kullanılan çevresel devrelere organize edilmiştir. Bir altdizi içindeki hücrelerin bir satırı (DRAM satırı), bir satır kodçözücü (*-ing*, row decoder) tarafından sürülen ve okunacak veya yazılacak hücre satırını açan bir tel (kelime hattı (*-ing*, wordline)) paylaşır. Bir altdizi içindeki hücrelerin bir sütunu (DRAM sütunu (*-ing*, DRAM column)), bir satır arabelleği (*-ing*, row buffer) (algılama amplifikatörlerinden (*-ing*, sense amplifier) oluşan) yardımıyla hücrelere okuma ve yazma işlemi yapmak için kullanılan bir tel (bit hattı (*-ing*, bitline)) paylaşır. Bir DRAM hücresi, elektrik yükü şeklinde bir bit bilgiyi depolayan bir depolama kapasitörü (*-ing*, storage capacitor) ve bir erişim transistöründen (*-ing*, access transistor) oluşur. DRAM hücrelerinin bu hiyerarşik düzeni, DRAM sistemindeki herhangi bir verinin ayrı sınıf, banka, satır ve sütun adresleri kullanılarak erişilmesini ve güncellenmesini sağlar.

DRAM işlemleri ve zamanlaması. Bir DRAM satırına erişmek için dört ana DRAM komutu vardır: *ACT*, *RD*, *WR* ve *PRE*. Şekil 2.2, bir DRAM hücresine erişmek için verilen komutlar ile bu komutların uyması gereken, JEDEC tarafından belirlenen, zamanlama parametreleri arasındaki ilişkiyi dört adımda göstermektedir. Başlangıçta, hücre kapasitörü VDD'yi depolar ve banka hazır durumdadır. ❶ Bellek kontrolcüsü *ACT* komutunu verir. *ACT* dört ana adımdan oluşur: 1) ilgili kelime hattını etkinleştirir, 2) dolayısıyla DRAM satırındaki tüm erişim transistörlerini etkinleştirir, 3) hücre kapasitörlerini ilgili bit hatlarına bağlar ve 4) her hücre kapasitörü ile ilgili bit hattı arasında analog bir yük paylaşım (*-ing*, charge sharing) süreci başlatır. ❷ Hücre ve bit hattı voltajları yük paylaşımı nedeniyle eşitlendiğinde, yük yenileme başlar. Yük yenileme sırasında, algılama amplifikatörleri önce bit hattı voltaj kaymasını tespit etmek ve ardından kaymanın yönüne bağlı olarak bit hattını VDD veya GND'ye geri yüklemek için etkinleştirilir. ❸ Hücre, erişime hazır bir voltaj seviyesine geri yüklendiğinde, ki bu *ACT* ve *RD*/*WR* komutları arasındaki zamanlama parametresi olan t_{RCD}'yi beklemeyi gerektirir, *RD* veya *WR* komutları bankaya verilebilir. ❹ Bellek kontrolcüsü, t_{RAS} tarafından belirtilen en az zaman aralığından sonra aynı bankaya *PRE* komutunu verebilir. t_{RAS}, etkinleştirilen satırın DRAM hücrelerinin ilk durumdaki voltaj seviyesine tamamen geri yüklenmesi için yeterli zamanın geçtiğini garanti eder. *PRE* gecikmesi, bit hattı voltajını referans voltajına (örneğin, $VDD/2$) geri ayarlamak için yeterli zamanı sağlayan t_{RP} zamanlama parametresi ile yönetilir. t_{RP}'den sonra, bellek kontrolcüsü aynı bankada yeni bir satırı açmak için *ACT* komutunu verebilir.



Şekil 2.2: Bir DRAM hücresinin yük yenileme sürecini gösteren zaman çizelgesi.

Yük yenileme işlemi. DRAM hücreleri, zamanla hücre kapasitöründen doğal olarak yük sızdırır. Yük sızdırması belirli bir seviyeyi aşarsa, hücrenin verisi algılama amplifikatörü tarafından doğru şekilde algılanamaz. Bu hatalar, veri tutma hataları olarak bilinir. Veri tutma hatalarını önlemek için bellek kontrolcüsü, DRAM hücrelerini yenilemek amacıyla *REF* komutları verir. Bir *REF* komutu, hücrelerin yük seviyesini algılayıp yeniden yüklemek için DRAM satırlarını içsel olarak etkinleştirir; bu işlem yaklaşık $35-40ns$ sürer [84–90]. Tek bir yenileme işlemi, bir satırı açmak (*ACT* komutu) ve bankayı kapatmak (*PRE* komutu) olarak düşünülebilir; bu Şekil 2.2'de gösterilmiştir. Veri bütünlüğünü korumak için bellek kontrolcüsü, her satırı belirli bir zaman aralığında (yenileme penceresi, t_{REFW} (*-ing*, refresh window)) periyodik olarak yeniler (DDR5 için $32ms$ [89] ve DDR4 için $64ms$ [88]). Tüm

satırların her t_{REFW} süresinde yenilendiğinden emin olmak için bellek kontrolcüsü, belirli bir zaman aralığıyla (yenileme aralığı, t_{REFI} (-*ing*, refresh interval)) REF komutları verir (DDR5 için $3,9\mu s$ [89] ve DDR4 için $7,8\mu s$ [88]). Her REF komutu, belirli bir zaman penceresi boyunca tüm DRAM bankasını veya sınıfını engeller; bu pencere yenileme gecikmesi (t_{RFC}) (-*ing*, refresh latency) olarak adlandırılır. t_{RFC}, yük yenileme gecikmesi ve bir REF komutuyla yenilenen satır sayısına bağlıdır.

2.2 DRAM Okuma Hataları

DRAM üretim teknolojisi düğüm boyutu küçüldükçe, satırlar arası etkileşim artar ve devre düzeyinde okuma hatası mekanizmalarına neden olur. Bu tür mekanizmalara iki önemli örnek, bir DRAM satırını (yani, saldırgan satır) tekrar tekrar etkinleştirmenin (yani, açmanın) veya saldırgan satırı uzun süre etkin tutmanın fiziksel olarak yakın satırlarda (yani, kurban satırlar) bit hatalarına neden olduğu SatırDarbesi [114] ve SatırBaskısı'dır (-*ing*, RowPress) [136]. SatırDarbesi zafiyeti bit hatalarını indüklemek için, bir saldırgan satırın SatırDarbesi eşiği olan N_{RH} değerinden daha fazla etkinleştirilmesi gerekir. Birçok karakterizasyon çalışması [2, 9, 36, 52, 56, 57, 72, 93, 101, 110, 114, 128–130, 136, 146–149, 153, 154, 156, 159, 161, 163–165, 177, 181, 213, 224, 227, 228, 230, 233, 242], DRAM teknolojisi daha küçük düğümlere doğru ilerledikçe, DRAM çiplerinin SatırDarbesi'ne karşı daha savunmasız hale geldiğini göstermektedir (yani, daha yeni çiplerin daha düşük N_{RH} değerleri vardır). Örneğin, 2020 yılında üretilen çiplerde SatırDarbesi bit hatalarını $4,8K$ etkinleştirme ile indüklemek mümkünken [110], 2013 yılında üretilen eski çiplerde bir satırın $69,2K$ kez etkinleştirilmesi gerekmektedir [114]. Durumu daha da kötüleştiren, hem SatırDarbesi hem de SatırBaskısı'nı kullanan erişim desenleri oluşturarak N_{RH} değerinin önemli ölçüde azaltılabilmesidir [136].

DRAM okuma hataları önleme mekanizmaları. Birçok önceki çalışma, farklı yaklaşımlar kullanarak DRAM çiplerini SatırDarbesi bit hatalarına karşı korumak için önleme mekanizmaları önermektedir [3–7, 11–14, 17, 23, 27, 42–44, 47, 49, 50, 52, 56, 58, 59, 63, 64, 66–68, 70, 73, 76, 78, 88, 94–97, 106, 112–114, 117, 123, 124, 126, 134, 135, 139–141, 149, 151, 152, 162, 166, 173, 174, 181, 185, 187–190, 193, 195, 198, 208, 211, 212, 217, 220, 223, 225–228, 232, 237, 239, 243, 244]. Bu mekanizmalar iki görevi yerine getirir: 1) bir tetikleyici algoritma yürütmek ve 2) önleyici bir eylem gerçekleştirmek. Tetikleyici algoritma, bellek erişim desenlerini gözlemler ve olasılıksal veya deterministik bir sürecin sonucuna göre önleyici bir eylemi tetikler. Önleyici eylem, 1) kurban satırları yenilemek, 2) saldırgan satırları dinamik olarak yeniden eşlemek veya 3) güvensiz erişimleri kısıtlamak olabilir.

DDR5 standardı, DRAM çiplerini SatırDarbesi saldırılarına karşı korumak için Yenileme Yönetimi (RFM) (-*ing*, Refresh Management) adı verilen yeni bir komut tanıtır [89]. Bellek kontrolcüsü, her DRAM bankasının etkinleştirme sayısını takip eder ve bu sayı belirli bir eşiği aştığında bir RFM komutu verir. DRAM modülü bir RFM komutu aldığında, belirli bir zaman penceresi içinde içsel olarak önleyici eylemler gerçekleştirir. Kaba taneli sayım nedeniyle, bellek denetleyicisi fazla miktarda RFM komutu verir ve bu da yüksek performans kayıplarına neden olur [26].

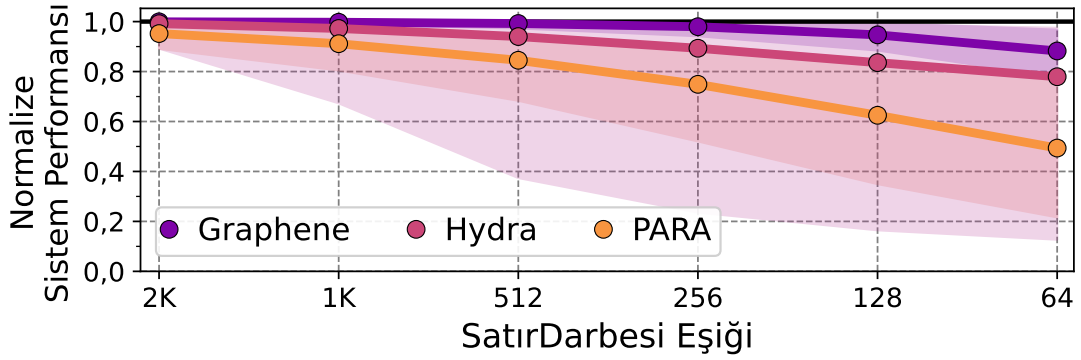
RFM komutlarının sayısını azaltmak için en son DDR5 standardı, Satır Başına Etkinleştirme Sayma (PRAC) (-*ing*, Per Row Activation Counting) adlı yeni bir mekanizmayı tanıtır [91]. PRAC, DRAM çipinin içine satırlar için etkinleştirme sayaçları yerleştirerek ince taneli sayımı sağlar. Bir satırın sayacı eşik değerini aştığında, DRAM modülü bellek denetleyicisinden bir RFM komutu istemek için yeni bir uyarı (-*ing*, back-off) sinyali gönderir. Bellek denetleyicisi bu sinyali aldığında, bir RFM komutu verir ve DRAM çipinin önleyici yenileme işlemleri gerçekleştirmesine izin verir.

Mevcut SatırDarbesi önleme mekanizmaları, tetikleyici algoritmaları daha agresif olacak şekilde konfigüre edildiğinde SatırBaskısı bit hatalarını da önleyebilir; bu, pratikte bunları $1K$ altı N_{RH} değerleri için konfigüre etmeye eşdeğerdir [136]. Ne yazık ki, mevcut SatırDarbesi önleme mekanizmaları, kendi önleyici eylemlerini daha agresif bir şekilde gerçekleştirdikleri için düşük N_{RH} değerlerinde (yani, $1K$ altı) çok büyük performans kayıplarına yol açar. Teknoloji düğüm boyutları küçüldükçe DRAM okuma hataları kötüleştiği göz önüne alındığında, N_{RH} değerlerinin daha da düşmesi beklenmektedir. Bu nedenle, mevcut SatırDarbesi savunmalarının performans kaybını azaltmak kritiktir.

3. MOTİVASYON

3.1 SatırDarbesi Önleme Mekanizmalarının Performans Kaybı

Önceki çalışmalar [110, 166, 226], mevcut SatırDarbesi önleme mekanizmalarının, üretim teknolojisi düğüm boyutu küçüldükçe ve dolayısıyla N_{RH} azaldıkça kabul edilemez derecede yüksek performans kayıplarına neden olduğunu zaten göstermektedir. Bu analizi, önceki analizlerden daha yeni olanlar da dahil olmak üzere üç son teknoloji SatırDarbesi önleme mekanizması ile tekrar ediyoruz. Bu simülasyonları, Bölüm 7.1'de belirtilen metodoji ve belirtilen test grupları (*-ing*, benchmark) arasından rastgele seçilen 62 tek çekirdekli test programı için 6 farklı N_{RH} değeri ile gerçekleştiriyoruz. Şekil 3.1, her eğrinin farklı bir SatırDarbesi önleme mekanizmasını temsil ettiği bir çizgi grafiği göstermektedir. Test edilen N_{RH} değerleri x-ekseni üzerindedir. Sistem performansı, çevrim-başına-buyruk cinsinden hiçbir önleme mekanizması kullanılmadığı değerine normalize olarak y-ekseni üzerinde gösterilmiştir. Taralı alanlar, tüm test programları arasındaki minimum ve maksimum değerlerin arasını göstermektedir.



Şekil 3.1: SatırDarbesi zafiyeti kötüleştikçe SatırDarbesi önleme mekanizmalarının artan performans kayıpları.

Şekil 3.1'den dört gözlemde bulunuyoruz. Birincisi, test edilen tüm SatırDarbesi önleme mekanizmaları için DRAM çipleri okuma hatalarına daha savunmasız hale geldikçe (yani, N_{RH} azaldıkça) sistem performansı önemli ölçüde azalır. İkincisi, test edilen SatırDarbesi önleme mekanizmaları arasında Graphene, tüm test edilen test programlarında N_{RH} değeri 64 olduğunda ortalama %6 performans kaybıyla en az performans kaybını sergiler. Graphene, en az performans kaybını sağlarken, her sınıfda 16 banka bulunan çift-sınıflı (*-ing*, dual-rank) bir sistem için konfigüre edildiğinde $10,38 mm^2$'ye ulaşan büyük bir çip alanı yükü getirir. Graphene, yüksek teknoloji bir Intel Xeon işlemcisinin çip alanının %4,7'sini tüketir [218]. Üçüncüsü, PARA neredeyse sıfır alan yükü getirirken, test edilen tüm N_{RH} değerleri için en yüksek performans kaybını sergiler ve N_{RH} 64 olduğunda test edilen test programlarında ortalama %31 yavaşlamaya ulaşır. Dördüncüsü, Hydra ve PARA'nın

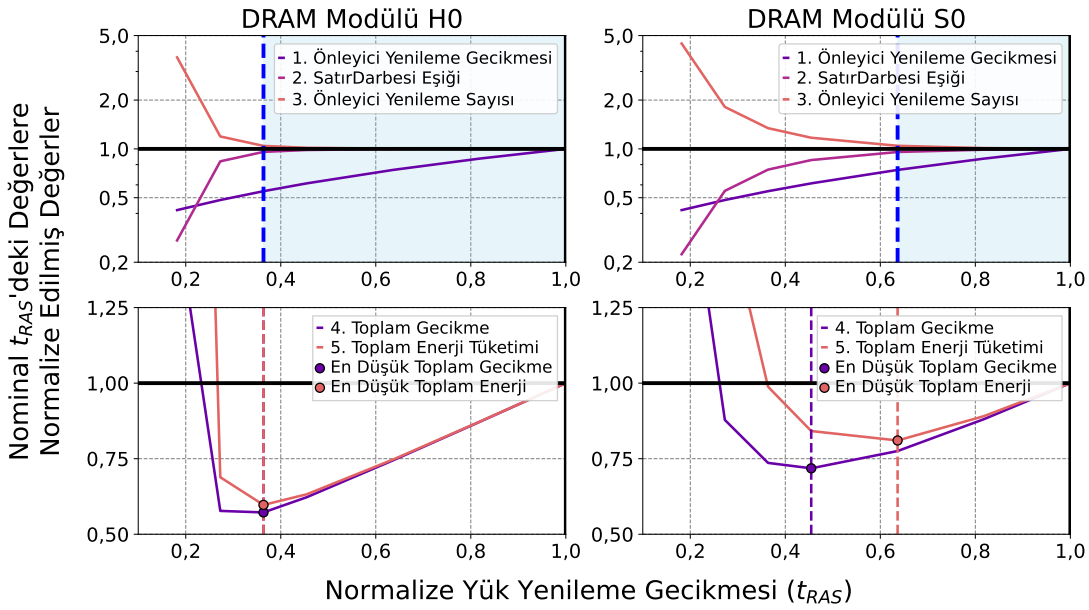
yavaşlamaları en kötü durumda sırasıyla %89 ve %79'a ulaşır. Bu dört gözlemle, mevcut SatırDarbesi önleme mekanizmalarının N_{RH} azaldıkça önemli performans veya çip alanı kayıplarına neden olduğunu gösteriyoruz. Bu nedenle, bu tür mekanizmaların performans kayıplarını düşük maliyetle azaltmanın kritik olduğu sonucuna varıyoruz.

3.2 SatırDarbesi Önleyici Yenileme İşlemleri için Harcanan Zaman ve Enerjiyi Azaltma

Önceki çalışmalar, DRAM zamanlama kısıtlamalarının büyük koruma bantları içerdiğini zaten göstermektedir [32–34, 107, 121, 224]. SatırDarbesi önleme mekanizmalarının performans kaybını azaltmak için, SatırDarbesi önleyici yük yenileme işlemleri için harcanan zamanı azaltma fikrini araştırıyoruz. Bu amaçla, bir satırın verilerini algılama ve bir DRAM hücresinin yükünü tamamen yenileme işlemi olan yük yenileme sürecinin zamanlamasını (t_{RAS}) azaltıyoruz. Bu, bir satır yenileme sırasında en fazla zaman alan işlemdir. t_{RAS}'ı azaltmanın etkisini 1) Bölüm 4'te açıklanan metodolojiyi kullanılarak gerçek DDR4 DRAM çiplerinin güvenilirliği üzerinde ve 2) önleyici yenileme işlemlerinin zaman ve enerji maliyeti üzerinde deneysel olarak değerlendiriyoruz.

Şekil 3.2, iki örnek DDR4 DRAM modülü için motivasyonel analizimizin sonuçlarını göstermektedir. Yük yenileme gecikmesini x-ekseni üzerinde sağdan sola doğru azaltıyoruz. $x = 1,0$, nominal t_{RAS} değerini işaretler. Y-ekseni, beş farklı eğrinin nominal t_{RAS} değerindeki kendi değerlerine normalize edilmiş değerlerini göstermektedir; bunların üçü üst iki grafikte ve kalan ikisi alt iki grafikte yer almaktadır. Numaralandırılmış beş eğri, 1) tek bir önleyici yenileme işlemini gerçekleştirmek için harcanan zamanı, 2) azalan yük yenileme gecikmesi kullanıldığında gerçek DRAM çiplerinde gözlemlenen ilk SatırDarbesi bit hatasını indüklemek için gereken minimum etkinleştirme sayısını (yani, N_{RH}), 3) N_{RH} ile ters orantılı olarak ölçeklendiği varsayılan tahmini önleyici yenileme işlemlerinin sayısını, 4) önleyici yenileme işlemleri sayısı ile bir yenileme işlemi için harcanan zamanın çarpımı olarak tüm SatırDarbesi önleyici yenileme işlemleri için harcanan toplam zamanı ve 5) gerçekleştirilen tüm önleyici yenileme işlemleri sayısı ile bu işlemleri gerçekleştirmek için harcanan toplam zamanın çarpımı olarak tahmini enerji tüketimini göstermektedir.

Şekil 3.2'den dört gözlemde bulunuyoruz. Birincisi, SatırDarbesi önleyici yenileme işlemi için harcanan zaman, yük yenileme gecikmesi azaldıkça orantılı olarak azalır. İkincisi, azalan yük yenileme gecikmesi 1) t_{RAS}, H0 ve S0 modülleri için sırasıyla nominal değerin %36 ve %64'üne düşene kadar N_{RH} üzerinde %5'ten fazla fark yaratmaz (dikey kesikli mavi çizgilerle işaretlenmiştir), ancak 2) daha düşük t_{RAS} değerleri SatırDarbesi eşiğinde önemli bir düşüşe neden olur. Üçüncüsü, tüm SatırDarbesi önleyici yenileme işlemleri için harcanan zaman, t_{RAS} azaldıkça ilk olarak azalır, ta ki H0 ve S0 modülleri için nominal t_{RAS} değerinin sırasıyla %36 ve %45'inde koyu mor bir daire ile işaretlenmiş bir kırılma noktasına kadar. Bu kırılma noktasının altındaki t_{RAS} değerleri için, gereken yenileme işlemlerinin sayısının artması, yenileme gecikmesindeki azalmayı bastırdığından, SatırDarbesi önleyici yenileme işlemleri için



Şekil 3.2: Yük yenileme gecikmesini azaltmanın SatırDarbesi önleyici yenileme işlemleri için harcanan zaman ve enerji üzerindeki etkisi.

harcanan toplam zaman artar. Bu kırılma noktasında, azalan t_{RAS}, H0 ve S0 modülleri için sırasıyla %43 ve %28 oranında SatırDarbesi önleyici yenileme işlemleri için harcanan toplam zamanı azaltır. Dördüncüsü, önleyici yenileme işlemlerinin enerji tüketimi, zaman maliyeti eğrisine benzer bir davranış izler ve kırılma noktaları, bu işlemlerin enerji tüketiminin sırasıyla %41 ve %19 azaltılabildiği nominal t_{RAS} değerinin %36 ve %64'ünde ortaya çıkar. Bu motivasyonel analize dayanarak, yük yenileme gecikmesini azaltmanın, SatırDarbesi önleme mekanizmalarının performans kaybını azaltmak için umut verici bir yaklaşım olduğunu sonucu varıyoruz.

3.3 Yük Yenileme Gecikmesinin Etkisini Karakterize Etmek

Önleyici yük yenileme işlemlerinin gecikmesini güvenilir bir şekilde azaltmak için bir DRAM çipinin sınırlarını anlamak kritiktir. DRAM çiplerinin güvenilirliği ve performansı üzerinde önemli etkileri olabilecek olan bu sınırların doğru bir şekilde belirlenmesi, etkili önleme mekanizmalarının geliştirilmesi açısından hayati öneme sahiptir. Birçok önceki çalışma [2, 9, 36, 52, 56, 57, 72, 93, 101, 110, 114, 128–130, 136, 146–149, 153, 154, 156, 159, 161, 163–165, 177, 181, 213, 224, 227, 228, 230, 233, 242], DRAM okuma hatalarının çeşitli yönlerini (örneğin, erişim/veri deseni, voltaj, sıcaklık) detaylı bir şekilde incelemiş ve bu hataların karakteristik özelliklerini belirlemiştir. Ancak, bu çalışmaların hiçbiri, yük yenileme gecikmesinin SatırDarbesi zafiyeti üzerindeki etkisini doğrudan ele almamıştır. Yük yenileme sürecinin zamanlamasını azaltmanın SatırDarbesi zafiyetine nasıl etki ettiğini ve bunun sonucunda sistem performansının nasıl değiştiğini anlamak, mevcut önleyici yenileme mekanizmalarının etkinliğini artırmak için önemlidir.

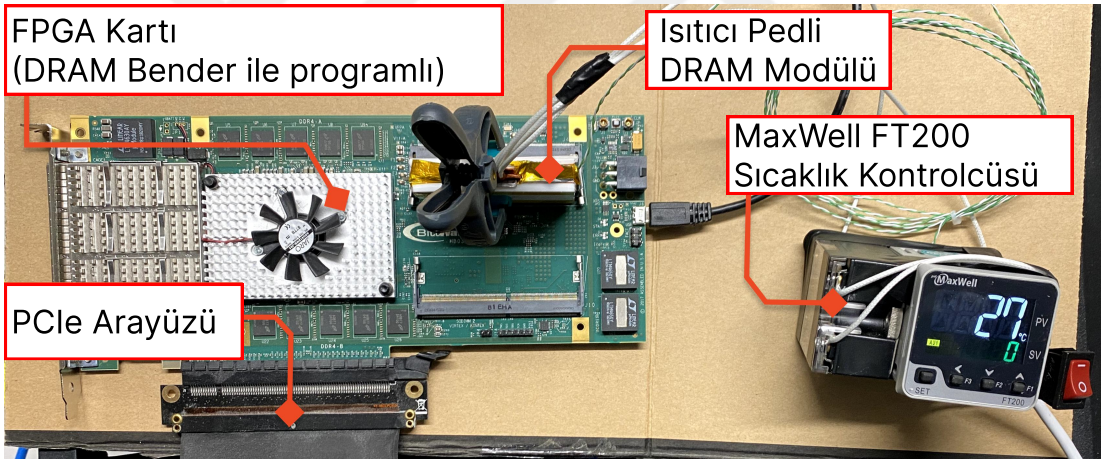
Tezin geri kalanında, yük yenileme gecikmesinin azaltılmasının gerçek DDR4 DRAM çiplerinin SatırDarbesi zafiyeti ve sistem performansı üzerindeki etkisini inceleyen kapsamlı bir deneysel karakterizasyon çalışması gerçekleştiriyoruz. Bu çalışmada, DRAM çiplerinin sınırlarını zorlayarak, yük yenileme süresinin azaltılmasının güvenilirlik ve performans üzerindeki etkilerini derinlemesine analiz edeceğiz.

4. METODOLOJİ

Bu bölüm, DRAM test altyapımızı, test edilen gerçek DDR4 DRAM çiplerini ve test metodolojimizi tanımlamaktadır.

4.1 DRAM Test Etme Altyapısı

DRAM Bender altyapısı. Modern DDR4 DRAM çiplerini, dört ana bileşenden oluşan FPGA tabanlı bir DRAM test altyapısı kullanarak test ediyoruz (Şekil 4.1'de gösterildiği gibi): 1) test programını oluşturan ve deney sonuçlarını toplayan bir ana makine, 2) test programlarımızı yürütmek üzere DRAM Bender [154, 184] (SoftMC [71, 183] tabanlı) ile programlanmış bir FPGA geliştirme kartı (Xilinx Alveo U200 [222]), 3) hedef sıcaklık seviyesini korumak için DRAM çiplerine karşı bastırılmış bir sıcaklık sensörü ve bir çift ısıtıcı ped ve 4) ısıtıcıları kontrol eden ve sıcaklığı $\pm 0.5\,^{\circ}C$ hassasiyetle istenen seviyede tutan[1] bir PID sıcaklık kontrol cihazı (MaxWell FT200 [143]).



Şekil 4.1: Karakterizasyonda kullanılan DRAM Bender altyapısı.

Müdahale kaynaklarını ortadan kaldırma. Devre/sistem düzeyinde müdahale olmadan SatırDarbesi bit hatalarını gözlemlemek için, önceki çalışmalarda olduğu gibi [70, 110, 136, 161, 224], potansiyel müdahale kaynaklarını ortadan kaldırmak için üç önlem alıyoruz. İlk olarak, test programlarımızın yürütülmesi sırasında periyodik yenilemeyi (*-ing*, periodic refresh) devre dışı bırakıyoruz, böylece periyodik yenilemeler veya potansiyel çip-üstünde TRR (*-ing*, on-die TRR) mekanizmaları [52, 70] tarafından istenmeyen yük yenilemesini önleyerek DRAM çipinin davranışını devre düzeyinde gözlemleyebiliyoruz. İkinci olarak, herhangi

[1] Sıcaklık kontrol cihazımızın SatırDarbesi testleri sırasında güvenilirliğini test etmek için, rastgele seçilmiş 5*K* DRAM satırında, dokuz farklı darbe sayısında SatırDarbesi testlerini tekrar ettik. Testlerin etkisiyle genel hata oranındaki değişimin $0.5\,^{\circ}C$'den az olduğunu gözlemledik.

bir programın çalışma süresinin yenileme penceresini aşmadığından emin oluyoruz, böylece veri tutma hataları gözlemlemiyoruz. Üçüncü olarak, test edilen DRAM modüllerinin ve çiplerinin ne sınıf düzeyinde ne de çip-üstünde ECC'ye sahip olmadığını doğruluyoruz. Bunları yapmak, mimari düzeydeki düzeltme ve hafifletme mekanizmalarından müdahale olmadan tüm devre düzeyinde bit hatalarını doğrudan gözlemlememizi ve analiz etmemizi sağlar.

4.2 Karakterize Edilen DDR4 DRAM Çipleri

Çizelge 4.1, üç büyük DRAM üreticisinden test ettiğimiz 30 modüldeki 388 gerçek DDR4 DRAM çipini göstermektedir. Analizimizin farklı DRAM teknolojilerine, tasarımlarına ve üretim süreçlerine uygulanabilir olup olmadığını araştırmak için, her DRAM çip üreticisinden farklı çip kapasiteleri ve çip revizyonlarına sahip çeşitli DRAM çiplerini test ediyoruz.[2] Deney süresini makul bir seviyede tutmak için, önceki çalışmalara benzer olarak [54, 55, 114, 136, 156, 161, 224, 225], her test edilen modülden tek bir bankanın toplamda $3K$ satırını kullanarak karakterizasyonumuzu gerçekleştiriyoruz; $1K$ başlangıçtan, $1K$ ortadan ve $1K$ DRAM bankasının sonundan olacak şekilde. Tüm sonuçlar, aksi belirtilmedikçe, üç üreticiden 388 DRAM çipinin $3K$ satırının verilerini raporlamaktadır.

Çizelge 4.1: Karakterize edilen DDR4 DRAM çipleri.

Çip Üreticisi	Modül IDs	Çip Sayısı (Modül Sayısı)	Çip Kapasitesi	Çip Revizyonu	Çip Org.	Üretim Tarihi
Ürt. H (SK Hynix)	H0	8(1)	4Gib	M	x8	N/A
	H1	8(1)	4Gib	X	x8	N/A
	H2	8(1)	4Gib	A	x8	N/A
	H3	32(1)	8Gib	M	x4	N/A
	H4-5	32(2)	8Gib	D	x8	2048
	H6	32(1)	8Gib	A	x4	N/A
	H7-8	32(2)	16Gib	C	x8	2136
Ürt. M (Micron)	M0-1-2	48(3)	8Gib	B	x4	N/A
	M3	16(1)	16Gib	F	x8	2237
	M4	4(1)	16Gib	E	x16	2046
	M5	32(1)	16Gib	E	x4	2014
	M6	4(1)	16Gib	B	x16	2126
Ürt. S (Samsung)	S0-1	32(2)	4Gib	F	x8	N/A
	S2-3-4	24(3)	4Gib	E	x8	1708
	S5	4(1)	4Gib	C	x16	N/A
	S6-7-8-9	32(4)	8Gib	D	x8	2110
	S10	16(1)	8Gib	C	x8	1809
	S11	8(1)	8Gib	B	x8	2052
	S12	8(1)	8Gib	A	x8	2212
	S13	8(1)	16Gib	B	x8	2315

[2]Bir DRAM çipinin teknoloji düğümü her zaman kamuya açık değildir. X çip revizyon kodu, çip revizyonu hakkında kamuya açık bilgi olmadığı anlamına gelir.

4.3 Karakterizasyon Metodolojisi

Ölçütler. Bir DRAM modülünün SatırDarbesi zafiyetini karakterize etmek için iki ölçüt kullanarak inceliyoruz: 1) N_{RH}, en az bir SatırDarbesi bit hatası oluşturmak için her saldırgan başına gereken minimum etkinleştirme sayısı ve 2) BER, bir satırdaki SatırDarbesi bit hatası yaşayan DRAM hücrelerinin oranı. Daha yüksek N_{RH}/BER, SatırDarbesi'ne karşı daha düşük/yüksek zafiyet gösterir.

Testler. Algoritma 1 ana test kodumuzu gösterir. Tüm testlerimizde çift taraflı darbeleme (*-ing*, double-sided hammering) [110, 114, 136, 161, 192] kullanılır; burada kurban satıra fiziksel olarak bitişik olan iki saldırgan satıra darbe yapılır. İki saldırgan satır, dönüşümlü olarak etkinleştirilir (Algoritma 1 satır 9-15). Bu bağlamda, bir darbe, iki saldırgan satırın birer defa etkinleştirilmesidir. Çift taraflı darbelemeyi, DDR4 komut zamanlama spesifikasyonları [88] dahilinde mümkün olan en yüksek etkinleştirme hızıyla gerçekleştiririz; çünkü bu erişim deseni, SatırDarbesi önleme mekanizmaları devre dışı bırakıldığında DRAM çiplerinde en etkili SatırDarbesi erişim deseni olarak belirtilmiştir [37, 52, 110, 114, 156, 161, 192]. SatırDarbesi zafiyeti ile kısmi yük yenileme arasındaki etkileşimi analiz etmek için, çift taraflı darbelemeyi gerçekleştirmeden, önce kurban satırlarda yük yenileme işlemi gerçekleştirilir. (Algoritma 1 satır 21). Kısmi yük yenileme fonksiyonu (Algoritma 1 satır 2-6), farklı yenileme gecikmeleri ve yenileme sayıları için kısmi yük yenileme gerçekleştirebilir. Farklı N_{RH} bulma yöntemlerini test etmek için, saldırgan satırlara darbe yapmadan önce veya sonra bekleme süreleri kullanırız (Algoritma 1 satır 22, 24). Azaltılmış yük yenileme gecikmesi nedeniyle herhangi bir hatanın oluşmamasını garanti etmek için, kurban satırları t_{REFW} süresince erişilmeden bırakırız (yani, periyodik yenileme nedeniyle yenilenene kadar) (Algoritma 1 satır 18-26). Bunu yaparak, azaltılmış yük yenileme gecikmesinin bir tutma hatasına neden olup olmadığını tespit edebiliriz. Örneğin, N_{RH} değeri 0, kurban satırın darbeleme yapılmadan tutma hatası nedeniyle bit hatası yaşadığını gösterir. Önceki çalışmalara benzer şekilde [110, 114, 120, 136, 156, 161, 224, 225], tüm deneylerimizi beş kere tekrar ederiz ve analizlerimizde en kötü durumu (yani, en düşük/en yüksek N_{RH}/BER) değerlerini kullanırız.

N_{RH} bulma. Satırların SatırDarbesi zafiyetini analiz etmek ve N_{RH} değerini ölçmek amacıyla basit bir ikili arama (*-ing*, binary search) algoritması uyguluyoruz (Algoritma 1 satır 37-43). Algoritma üç parametre belirler: $HChigh$, $HClow$ ve $HCstep$. $HChigh$/$HClow$, N_{RH} için üst/alt sınırı belirler ve $HCstep$, N_{RH} ölçümünün çözünürlüğünü belirler. Tüm DDR4 DRAM çiplerimizi $HChigh$/$HClow$ için $100K$/0 darbe ve $HCstep$ için $1K$ ile test ediyoruz.

Veri desenleri. Çizelge 4.2, testlerde kullanılan altı veri desenini gösterir. Altı yaygın kullanılan veri desenini kullanıyoruz [33, 34, 102–104, 110, 114, 120, 132, 144, 161]: satır şeridi, dama deseni, sütun şeridi ve bunların tersleri. Bir satırın N_{RH} değerini ölçmeden önce, her satır için altı veri deseni arasından en etkili veri desenini (WCDP) belirleriz (Algoritma 1 satır 33-36). O satırdaki en fazla bit hatasına neden olan veri desenini kullanarak o satırın N_{RH} değerini ölçeriz.

Algoritma 1: Azaltılmış t_{RAS} değerinin SatırDarbesi üzerindeki etkisini gözlemlemek için ana test kodu.

```
// t_RAS(Nom): Nominal yük yenileme gecikmesi
// t_RAS(Red): Azaltılmış yük yenileme gecikmesi
// RA_victim: Kurban satır adresleri
// TestedRows: Test edilen satırların listesi (Bölüm 4.2)
// N_PR: Ardışık kısmi yük yenileme sayısı
// TestedN_PR: Test edilen N_PR listesi (Bölüm 5.3)
// DP: Veri deseni
// N_RH_method: Önce_Uyku/Önce_Darbe (Bölüm 5.1)
// HC: Saldırgan satır başına darbeleme sayısı

Function partial_restoration(RA_victim, t_RAS(Red), N_PR):
    for i = 0; i < N_PR; i++ do
        ACT(RA_victim, wait=t_RAS(Red))
        PRE(wait=t_RP)
    end


Function hammer_doublesided(RA_victim, HC):
    for i = 0; i < HC; i++ do
        ACT(RA_victim + 1, wait=t_RAS(Nom))
        PRE(wait=t_RP)
        ACT(RA_victim − 1, wait=t_RAS(Nom))
        PRE(wait=t_RP)
    end


Function measure_BER(RA_victim, DP, HC, t_RAS(Red), N_PR, N_RH_method):
    initialize_row (RA_victim, DP)
    initialize_aggressor_rows (RA_victim, DP)
    partial_restoration(RA_victim, t_RAS(Red), N_PR)
    if N_RH_method = Sleep_first then sleep()
    hammer_doublesided (RA_victim, HC)
    if N_RH_method = Hammer_first then sleep()
    BER_row = check (RA_victim)
    return BER_row


Function test_loop():
    foreach t_RAS(Red) in from t_RAS(Nom) to 6ns at steps of 3ns do
        foreach N_PR in from TestedN_PR do
            foreach RA_victim in TestedRows do
                // Find the worst-case data pattern
                foreach DP in [RS, RSI, CS, CSI, CB, CBI] do
                    measure_BER(RA_victim, DP, 100K, t_RAS(Red), N_PR)
                    WCDP = DP that causes largest BER
                end
                // Binary search for N_RH
                HChigh = 100K; HClow = 0; HCstep = 1K;
                while HChigh − HClow > HCstep do
                    HCcur = (HChigh + HClow)/2
                    b = measure_BER(RA_victim, WCDP, HCcur, t_RAS(Red), N_PR)
                    HClow = (b == 0)?HCcur : HClow
                    HChigh = (b! = 0)?HCcur : HChigh
                end
            end
        end
    end
```

Çizelge 4.2: Karakterizasyonda kullanılan veri desenleri.

Veri Deseni	**Saldırgan Satırlar**	**Kurban Satır**
Satır Şeridi	$0xFF$	$0x00$
Ters Satır Şeridi	$0x00$	$0xFF$
Sütun Şeridi	$0xAA$	$0xAA$
Ters Sütun Şeridi	$0x55$	$0x55$
Dama Deseni	$0xAA$	$0x55$
Ters Dama Deseni	$0x55$	$0xAA$

Fiziksel olarak bitişik satırları bulma. DRAM üreticileri, DRAM arayüzü üzerinden bellek kontrolcüsüne sunulan mantıksal DRAM adreslerini (örneğin, satır, banka ve sütun), fiziksel DRAM adreslerine (örneğin, bir satırın fiziksel konumu) çevirmek için DRAM içi adres haritalama şemaları kullanır [37, 115]. Dahili adres haritalama şemaları, 1) üretim sonrası satır onarım tekniklerinin hatalı DRAM satırlarını yedek satırlara yeniden eşleyerek onarmasına ve 2) üreticilerin DRAM iç yapısını maliyet açısından optimize edilmiş bir şekilde organize etmelerine olanak tanır. Haritalama şeması, farklı DRAM çipleri arasında önemli ölçüde değişebilir [16, 37, 77, 80, 100, 103, 105, 114, 120, 132, 161, 167, 168, 188]. Test ettiğimiz her kurban DRAM satırı için, bellek kontrolcüsünün çift taraflı bir SatırDarbesi saldırısında saldırgan satırlara erişmek için kullanabileceği iki komşu fiziksel olarak bitişik DRAM satır adresini belirliyoruz. Bunu yapmak için, önceki çalışmalarda açıklanan teknikleri kullanarak fiziksel satır organizasyonunu tersine mühendislik ile inceliyoruz [110, 136, 161].

Sıcaklık. Tüm testlerimizi 50 °C, 65 °C ve 80 °C olmak üzere üç sıcaklık seviyesinde gerçekleştiriyoruz. 80 °C, yenileme penceresinin yarıya indiği 85 °C'ye 5 °C'lik küçük ama güvenli bir marj bırakmak için bizim durumumuzda mümkün olan en yüksek sıcaklıktır. Aksi belirtilmedikçe karakterizasyon sonuçlarımızı 80 °C için rapor ediyoruz.

5. KARAKTERİZASYON

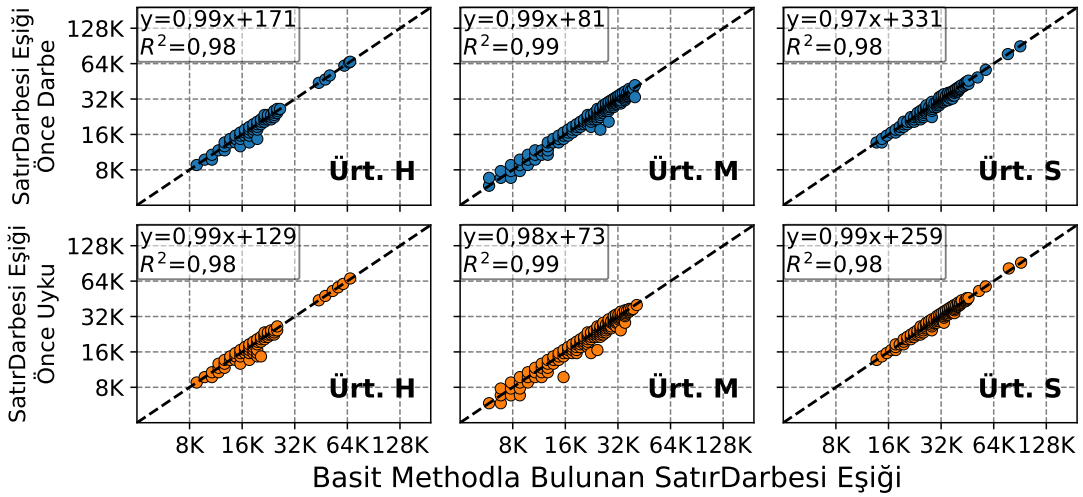
Bu bölümde, veri tutma süresi, yük yenileme ve SatırDarbesi zafiyeti arasındaki etkileşimin ilk kapsamlı karakterizasyonunu dört deney seti kullanarak sunuyoruz. Birincisi, bir DRAM çipinin SatırDarbesi zafiyetini doğru bir şekilde ölçmek için veri tutma süresi ile SatırDarbesi arasındaki ilişkiyi analiz ediyoruz. İkincisi, doğru yöntemimizi kullanarak yük yenileme gecikmesinin SatırDarbesi zafiyeti üzerindeki etkisini analiz ediyoruz. Üçüncüsü, tekrarlanan azaltılmış yük yenileme gecikmesinin etkisini analiz ediyoruz. Dördüncüsü, yük yenileme gecikmesinin veri tutma süreleri üzerindeki etkisini analiz ediyoruz.

5.1 SatırDarbesi ve Tutma Hataları

Karakterizasyon çalışmamızın ilk adımı olarak, Algoritma 1'deki metodolojiyi takip ederek veri tutma süresinin SatırDarbesi karakterizasyon sonuçları üzerindeki etkisini araştırıyoruz. Hipotezimiz, SatırDarbesi'nin bir sonucu olarak artan yük sızdırmasının, bir DRAM hücresinin yalnızca darbe nedeniyle değil, hücrenin doğuştan gelen sızdırma özellikleri ve SatırDarbesi'nin etkisinin birleşimi nedeniyle bit hatası yaşamasına yol açabileceğidir. Veri tutma süresi ile SatırDarbesi arasındaki bu ilişki doğal görünse de, önceki çalışmalar bu yönü araştırmamıştır. İlk olarak, kurban satırlar başlatıldıktan hemen sonra saldırgan satırların darbelendiği ve saldırgan satırlar darbelendikten hemen sonra kurban satırların okunduğu (yani, kurban satırlar yalnızca SatırDarbesi saldırısı sırasında erişilmeden kalır) basit bir yöntem kullanarak temel N_{RH}'yi ölçüyoruz. Daha sonra, iki farklı yöntem kullanıyoruz ve bunları basit yöntemle bulunanN_{RH} değerleri ile karşılaştırıyoruz. Testlerimiz, kurban satırını tam olarak bir t_{REFW} ($64ms$) boyunca erişilmeden tutar ve darbelemeyi başlangıçta (Önce Darbe) veya sonunda (Önce Uyku) gerçekleştirir. İstenilen darbe sayısına bağlı olarak saldırgan satırların darbelendirilmesi için gereken toplam süreyi ve t_{REFW}'ye kadar kalan süreyi hesaplarız.

Şekil 5.1, her üreticiden birer örnek modülden test edilen satırların N_{RH} değerlerindeki değişimi, iki farklı SatırDarbesi test yöntemi için göstermektedir (alt ve üst grafikler). X-ekseni, basit yöntem kullanılarak bulunan N_{RH} değerini ve y-ekseni, Önce Darbe veya Önce Uyku yaklaşımları kullanılarak bulunan N_{RH} değerlerini gösterir. Kesikli çizgi, her iki yöntemin de aynı N_{RH} değerini önerdiği $x = y$ çizgisini gösterir. Kesikli çizginin altındaki bir işaret, basit yöntemin, y-ekseni üzerinde belirtilen diğer yönteme göre daha yüksek darbe sayılarında bit hatalarına neden olduğu bir satırı gösterir, bu da SatırDarbesi önleme mekanizmaları için yanıltıcı olabilir. Her grafik için, verileri doğrusal regresyon modeline uydururuz ve bu modelin R^2 skoru ile birlikte her grafiğin sol üst köşesinde belirtiyoruz.

Şekil 5.1'den üç gözlemde bulunuyoruz. Birincisi, Önce Darbe ve Önce Uyku yöntemlerini kullanarak bulunan N_{RH} değerleri, çoğu zaman basit yöntem kullanarak



Şekil 5.1: Önce Darbe (üst grafikler) ve Önce Uyku (alt grafikler) yöntemleri ile bulunan N_{RH} değerlerinin basit methodla bulunan N_{RH} değerlerine göre değişimi.

bulunan N_{RH} değerlerinden daha düşük ya da eşittir. Test edilen satırların $\%57,93$'ü ve $\%44,05$'i, Önce Darbe ve Önce Uyku yöntemleriyle test edildiğinde basit yönteme göre daha düşük N_{RH} değerleri sergiler. Bu nedenle, hem Önce Darbe hem de Önce Uyku yöntemleri, en kötü durum N_{RH} değerini bulmada basit yöntemden daha doğrudur. İkincisi, farklı yöntemler kullanılarak bulunan N_{RH} değerleri birbirine benzerdir, çünkü veri noktalarının çoğu $y = x$ çizgisi etrafında toplanmıştır ve doğrusal regresyon modelleri $y = x$'e ($y = mx + b$ olan yerde $0,96 \leq m \leq 1,00$) yüksek güvenle ($R^2 \geq 0,98$) yaklaşır. Üçüncüsü, Önce Darbe ve Önce Uyku yöntemleri, çoğu satır için aynı N_{RH} değerini verirken, Önce Darbe'nin N_{RH} değeri, satırların $\%45,74$'ü için Önce Uyku'ya göre daha küçük (daha kötü) çıkmaktadır. Bu gözlemlere dayanarak, Çıkarım 1'i türetiyoruz.

Çıkarım 1. Önce Darbe, test edilen yöntemler arasında en kötü durum N_{RH} değerini bulmak için daha doğru bir yöntemdir.

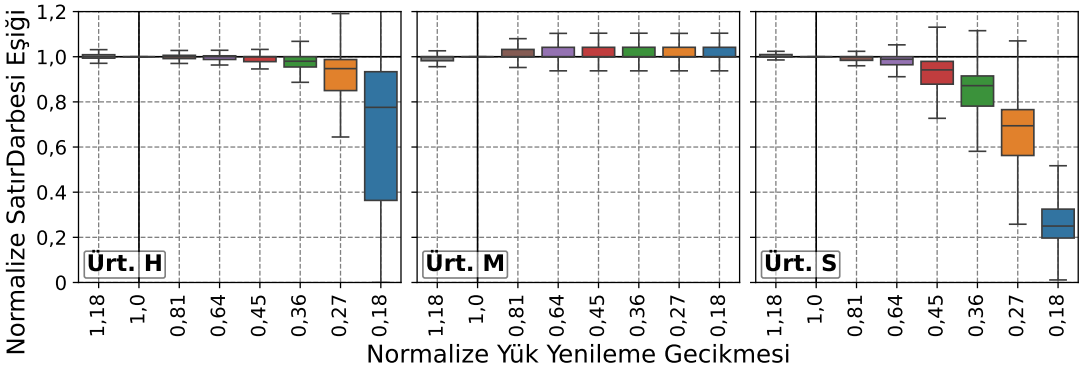
En kötü durum N_{RH} değerini ölçtüğü için karakterizasyon çalışmalarımıza Önce Darbe yöntemi ile devam ediyoruz.

5.2 SatırDarbesi ve Yük Yenilemesi

Bu bölüm, Bölüm 4'te açıklanan metodolojiyi takip ederek, gerçek DDR4 DRAM çiplerinde yük yenileme gecikmesinin SatırDarbesi zafiyeti üzerindeki etkilerinin ilk kapsamlı karakterizasyonunu sunmaktadır. Algoritma 1'de açıklandığı gibi, kurban satırının yükünü yenilemek için kullanılan yük yenileme gecikmesini değiştirirken DRAM satırlarının N_{RH} değerini ölçüyoruz.

Şekil 5.2, kutu grafiğinde yük yenileme gecikmesi azaltıldıkça N_{RH}'deki değişimi

göstermektedir.[3] X-ekseni, darbelemeden önce kurban satırları yenilemek için kullanılan normalize yük yenileme gecikmesini ve y-ekseni, kurban satırının darbelemeden önce nominal yük yenileme gecikmesi kullanılarak yenilendiğinde N_{RH} değerine normalize edilmiş N_{RH} değerlerini gösterir.



Şekil 5.2: Kısmi yük yenileme uygulandığında N_{RH} değerlerinin dağılımı.

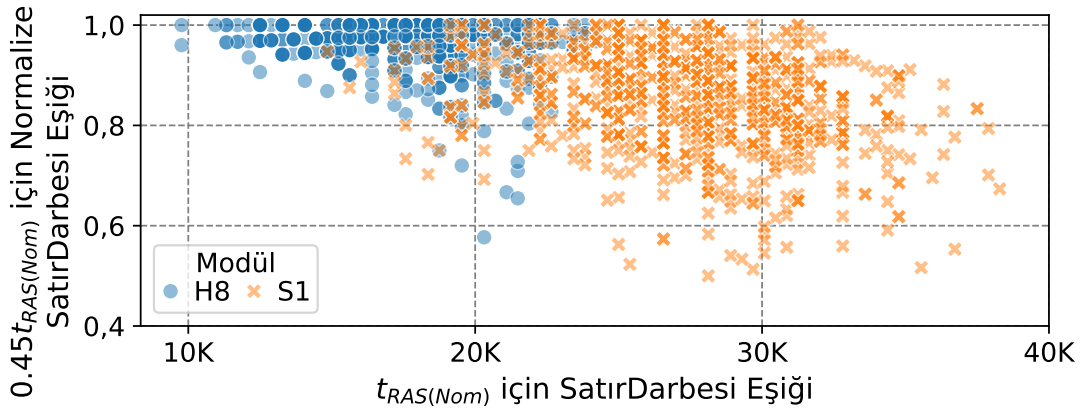
Şekil 5.2'den üç gözlemde bulunuyoruz. Birincisi, Ürt. H ve Ürt. S için yük yenileme gecikmesi azaldıkça test edilen satırların N_{RH} değeri önemli ölçüde düşer. İkincisi, çok düşük t_{RAS} değerlerinde (örneğin, nominal t_{RAS}'in %18'i), bazı DRAM hücreleri için yük yenileme yeterince güçlü değildir ve bu nedenle, bu hücreler darbe yapılmadan bit hataları yaşar ve N_{RH} değeri 0 olarak gözlemlenir. Üçüncüsü, yük yenileme gecikmesinin %36 ve %82 oranında azaltılması, sırasıyla Ürt. H ve Ürt. M'den test edilen satırların N_{RH} değerini etkilemez. Bu gözleme dayanarak, Ürt. H ve Ürt. M'nin, SatırDarbesi'ni kötüleştirmeden önemli ölçüde azaltılabilecek büyük bir koruma bandı uyguladığını varsayıyoruz.

N_{RH} azalım dağılımını analiz etmek için, satırların normalize N_{RH} değerleri ile nominal gecikme uygulandığında ölçülen (t_{RAS}) N_{RH} değerlerini inceliyoruz. Bu analizi, sadece t_{RAS} değeri azaltıldığında artan bir SatırDarbesi zafiyeti yaşayan Ürt. H ve Ürt. S'den gelen DRAM çipleri üzerinde gerçekleştiriyoruz.

Şekil 5.3, Ürt. H ve Ürt. S'den iki örnek modül için yük yenileme gecikmesi azaltıldıkça N_{RH}'deki değişimin dağılımını göstermektedir. X ekseni, kurban satırının darbelemeden önce azaltılmış yük yenileme gecikmesi ($0,45t_{RAS}$) kullanılarak yenilendiğindeki N_{RH} değerlerinin nominal yük yenileme gecikmesi (t_{RAS}) kullanılarak ölçülen N_{RH} değerlerine oranını gösterir. Y ekseni, her satır için nominal yenileme gecikmesi ile N_{RH} değerlerini gösterir. Her işaretçi stili farklı bir modülü temsil ederken, renkler üreticileri temsil eder.

Şekil 5.3'den iki gözlemde bulunuyoruz. Birincisi, satırların küçük bir kısmı, satırların büyük çoğunluğuna göre kısmi yük yenilemesine çok daha hassastır. Örneğin, test edilen satırların yalnızca $\%0,45/\%10,34$'ü Ürt. H/S için $0,75$'ten

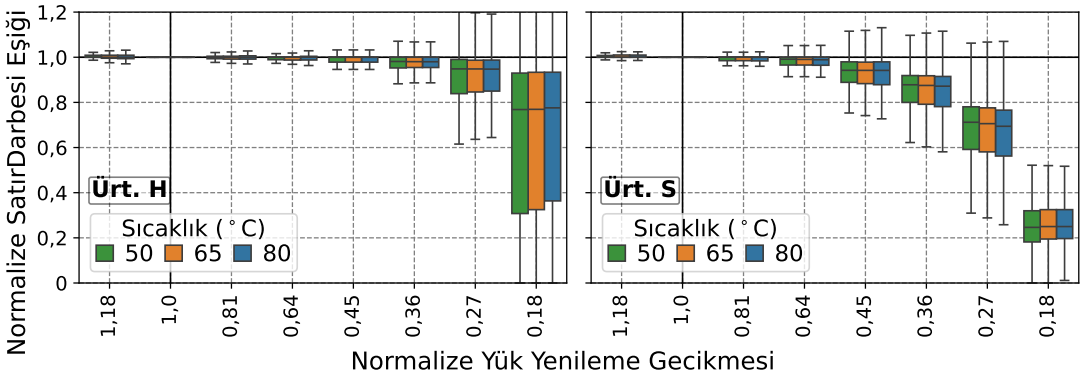
[3] Kutu, alt sınırı birinci çeyrek (yani, sıralı veri noktaları setinin ilk yarısının medyanı) ve üst sınırı üçüncü çeyrek (yani, sıralı veri noktaları setinin ikinci yarısının medyanı) tarafından belirlenmiştir. Çeyrekler arası aralık, birinci ve üçüncü çeyrekler arasındaki mesafedir (yani, kutu boyutu). Çizgiler ise minimum ve maksimum değerleri gösterir.



Şekil 5.3: $0.45t_{RAS(Nom)}$ gecikme kullanıldığındaki N_{RH} değerlerindeki düşüşün $t_{RAS(Nom)}$ gecikme kullanıldığındaki N_{RH} değerlerine göre dağılımı.

az N_{RH} değer düşüşüne sahiptir. İkincisi, kurban satırları nominal yük yenileme gecikmesi kullanılarak yenilendiğinde, en düşük N_{RH} değeri en kötü N_{RH} azalma oranını yaşamaz. Örneğin, kurban satırları nominal yenileme gecikmesi kullanılarak yenilendiğinde en düşük %10 N_{RH} değerine sahip olan satırların en kötü N_{RH} azalma oranı Ürt. H/S için $\%4,00/\%12,50$'dir.

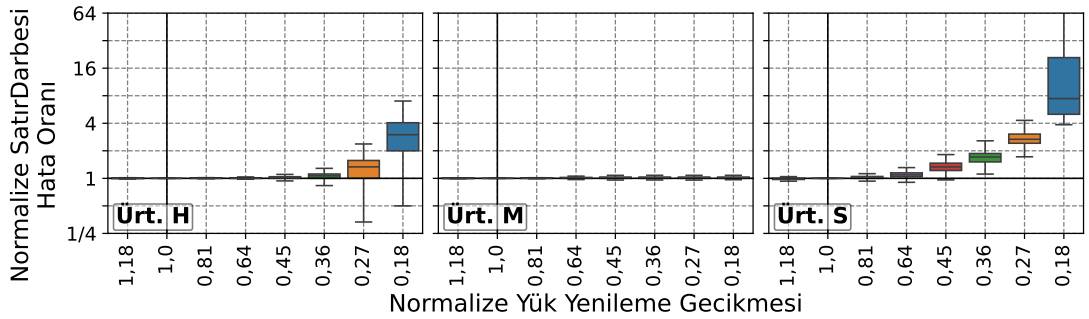
Sonraki olarak, sıcaklık ve yenileme gecikmesinin SatırDarbesi zafiyeti üzerindeki etkisini inceliyoruz. Şekil 5.4, kutu grafiğinde yük yenileme gecikmesi azaltıldıkça N_{RH}'deki değişimi göstermektedir.[3] X-ekseni, darbelemeden önce kurban satırları yenilemek için kullanılan normalize yenileme gecikmesini ve y-ekseni, kurban satırının darbelemeden önce nominal yenileme gecikmesi kullanılarak yenilendiğindeki N_{RH} değerine normalize edilmiş N_{RH} değerini gösterir. Üç renk, farklı sıcaklık değerlerini temsil etmektedir.



Şekil 5.4: Üç sıcaklık seviyesi için kısmi yük yenileme uygulandığında N_{RH} değerlerinin dağılımı.

Şekil 5.4'ten bir önemli gözlemde bulunuyoruz. Sıcaklık, SatırDarbesi zafiyeti dağılımını önemli ölçüde etkilemez. Örneğin, sıcaklık 50 °C'den 80 °C'ye yükseltildiğinde, Ürt. H/S'den gelen çipler için normalize N_{RH} sırasıyla $\%0,31/\%0,08$ oranında değişir.

Şekil 5.5, kutu grafiğinde yük yenileme gecikmesi azaltıldıkça SatırDarbesi *BER*'indeki değişimi göstermektedir.[3] X-ekseni, darbelemeden önce kurban satırları yenilemek için kullanılan normalize yük yenileme gecikmesini ve y-ekseni, kurban satırının darbelemeden önce nominal yük yenileme gecikmesi kullanılarak yenilendiğindeki *BER* değerine normalize edilmiş *BER* değerlerini gösterir.



Şekil 5.5: Kısmi yük yenileme uygulandığında *BER* değerlerinin dağılımı.

Şekil 5.5'ten iki gözlemde bulunuyoruz. Birincisi, Ürt. H ve Ürt. S için test edilen satırların *BER* değerleri, yük yenileme gecikmesi azaldıkça süperlineer şekilde artar. İkincisi, yük yenileme gecikmesinin %36 ve %82 oranında azaltılması, sırasıyla Ürt. H ve Ürt. M'den test edilen satırların *BER* değerini önemli ölçüde artırmaz. Bu gözlemler, Ürt. H ve Ürt. M'nin nominal yük yenileme gecikmesi kısıtlamalarında büyük bir koruma bandı uyguladığı ve bu bandın SatırDarbesi zafiyetini artırmadan önemli ölçüde azaltılabileceği hipotezimizi destekler. Bu gözlemlerden yola çıkarak Çıkarım 2'yi türetiyoruz.

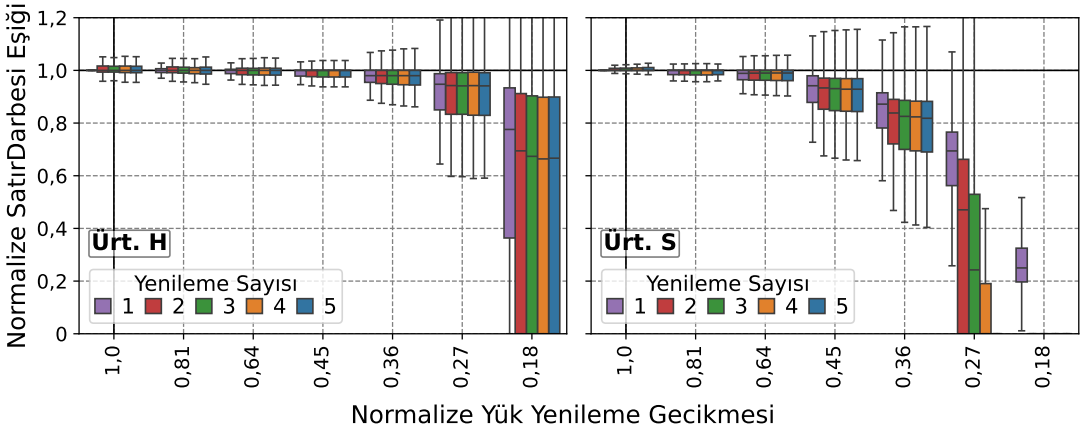
Çıkarım 2. Yük yenileme gecikmesi (t_{RAS}) 1) herhangi bir etki olmadan veya 2) gerçek DRAM çiplerinin SatırDarbesi eşiğini azaltma pahasına azaltılabilirken, yük yenileme gecikmesinin bir eşik değerinin ötesine indirilmesi veri tutma bit hatalarına neden olabilir ve güvenilirliği tehlikeye atabilir.

5.3 Tekrarlı Kısmi Yük Yenileme

Yük yenileme gecikmesinin azaltılması, DRAM hücrelerindeki yükün kısmen yenilenmesine yol açabilir. Bu kısmi yük yenileme işleminin tekrarlanması, sonunda depolanan yükü, yük sızdırma mekanizmalarına karşı yeterince güçlü olmayan bir seviyeye düşürebilir. Bu bölüm, Algoritma 1'deki metodolojiyi takip ederek, kısmi yük yenilemenin tekrarlanmasının SatırDarbesi zafiyeti üzerindeki etkisini araştırmaktadır. Bu analizi, yalnızca t_{RAS} değeri azaltıldığında artan bir SatırDarbesi zafiyeti yaşayan Ürt. H ve Ürt. S'den gelen DRAM çipleri üzerinde gerçekleştiriyoruz.

Şekil 5.6, kutu grafiğinde, kurban satır, darbeleme testinden önce tekrar tekrar yenilendiğinde ve yük yenileme gecikmesi azaltıldığında N_{RH}'deki değişimi göstermektedir.[3] X-ekseni, darbelemeden önce kurban satırları yenilemek için kullanılan normalize yük yenileme gecikmesini ve y-ekseni, kurban satırının darbelemeden önce nominal yük yenileme gecikmesi kullanılarak yenilendiğindeki

N_{RH} değerine normalize edilmiş N_{RH} değerini gösterir. Farklı renkler (açıklama kısmında belirtildiği gibi), kurban satır üzerinde SatırDarbesi saldırısından önce azaltılmış yük yenileme gecikmesi ile gerçekleştirilen yenileme sayısını gösterir.

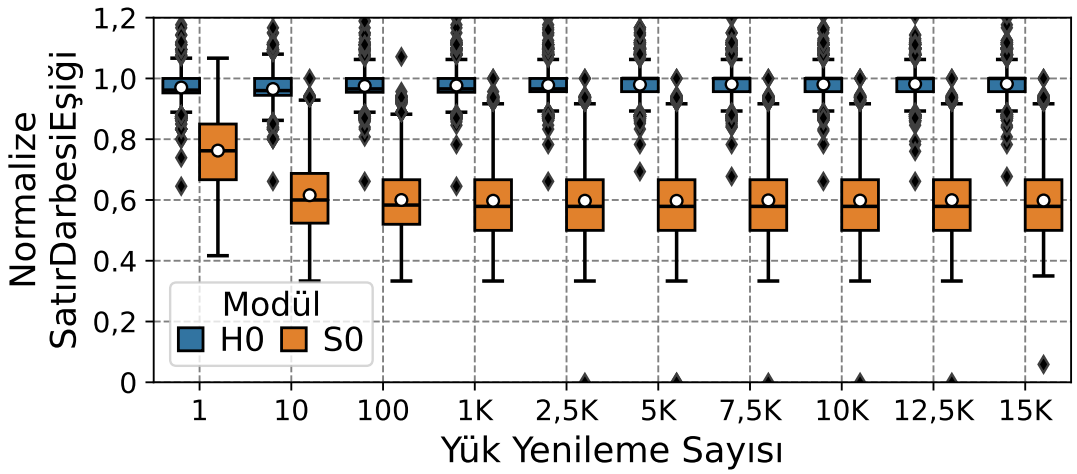


Şekil 5.6: Tekrarlı kısmi yük yenileme uygulandığında N_{RH} değerlerinin dağılımı.

Şekil 5.6'dan üç gözlemde bulunuyoruz. Birincisi, Ürt. H'den test edilen satırların N_{RH} değeri, restorasyon sayısına göre önemli ölçüde değişmez çünkü dağılım tüm tekrarlar boyunca benzerdir. İkincisi, Ürt. S'den test edilen satırların N_{RH} değeri, restorasyon sayısı arttıkça azalır. Örneğin, $t_{RAS(Red)} = 0,36t_{RAS(Nom)}$ olduğunda dağılımlar arasında belirgin bir azalma eğilimi vardır. Üçüncüsü, kısmi yenilemenin birçok kez gerçekleştirilmesi, yenileme gecikmesindeki azalmaya bağlı olarak tutma bit hatalarına neden olabilir. Örneğin, $t_{RAS(Red)} = 0,27t_{RAS(Nom)}$ kullanılarak yapılan tek bir kısmi yük yenilemesi, Ürt. S'den test edilen satırlar için veri tutma bit hatalarına neden olmaz. Ancak, bu işlemin iki kez tekrarlanması, bazı satırların N_{RH} değerinin sıfır olmasına, yani darbeleme olmadan bit hataları yaşamalarına yol açabilir.

Tekrarlanan yük yenilemenin etkisini daha detaylı bir şekilde araştırmak için, örnek iki modülü $t_{RAS(Red)} = 0,36t_{RAS(Nom)}$ için tekrar sayısını $15K$'ya kadar artırarak test ediyoruz. Şekil 5.7, kutu grafiğinde, kurban satır, darbeleme testinden önce azaltılmış yük yenileme gecikmesi ile tekrar tekrar yenilendiğinde N_{RH}'deki değişimi göstermektedir.[3] X-ekseni, kurban satır üzerinde darbeleme testinden önce, $0,36t_{RAS(Nom)}$ azaltılmış yenileme gecikmesi ile gerçekleştirilen yenileme sayısını gösterir. Y-ekseni, kurban satırının darbelemeden önce nominal yenileme gecikmesi kullanılarak yenilendiğindeki N_{RH} değerine normalize edilmiş N_{RH} değerini gösterir. Farklı renkler, farklı DRAM modüllerini temsil eder.

Şekil 5.7'den iki gözlemde bulunuyoruz. Birincisi, Ürt. H'den gelen satırlar, kısmi yük yenileme sayısından önemli ölçüde etkilenmez. Örneğin, 1 yenileme sonrası ölçülen N_{RH} değerleri, Ürt. H için azaltılmış yük yenileme gecikmesi ile $15K$ kez yenilendiğinde benzer bir varyasyon gösterir. İkincisi, birçok kez azaltılmış yük yenileme gecikmesi ile tekrarlanan yenilemeler, Ürt. S'den gelen satırların 2500 kez azaltılmış gecikme ile yenilendikten sonra darbeleme olmadan bit hataları yaşamasıyla sonuçlanabilir (yani, 2500 kısmi yük yenileme gerçekleştirmek hücrelerin yükünü kaybetmesine ve tutma hataları göstermesine neden olur). Bu nedenle, tekrarlanan



Şekil 5.7: $0,36t_{RAS(Nom)}$ gecikme ile tekrarlı kısmi yük yenileme uygulandığında N_{RH} değerlerinin dağılımı.

kısmi yük yenileme nedeniyle bit hatalarını önlemek için, Ürt. H/S'den gelen hücreler, azaltılmış gecikme ile $15K/1K$'dan fazla yenilenmemelidir. Bu gözlemlerden yola çıkarak Çıkarım 3'ü türetiyoruz.

Çıkarım 3. Tüm yenileme işlemleri için yük yenileme gecikmesini (t_{RAS}) azaltmak güvenli değildir. Ürt. H ve S'den gelen bir DRAM satırı, azaltılmış t_{RAS} ile yapılan her $15K$ ve $1K$ yenilemeden sonra nominal yük yenileme gecikmesi kullanılarak yenilenmelidir.

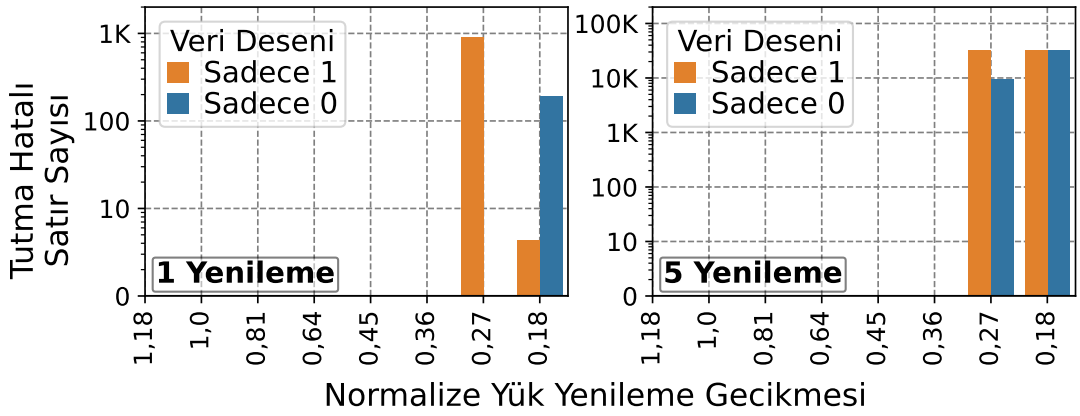
5.4 Veri Tutma Hatası ve Yük Yenileme

Bu bölümde, t_{RAS} değerinin azaltılmasının, bir DRAM çipinin $64ms$'lik nominal yenileme penceresinde yenilendiğinde tutma bit hatalarına yol açıp açmadığını test ediyoruz. Her üreticiden iki modül seçiyoruz ve her modülden bir bankta $32K$ satırı, $80\,^{\circ}C$'de iki veri deseni kullanarak test ediyoruz. Bu bit hatalarını tespit etmek için, satırları sadece *mantık-1* veya *mantık-0* ile başlatıyoruz ve satırı azaltılmış yenileme gecikmesi ile bir veya beş kez yeniliyoruz. $64ms$ bekliyor ve herhangi bir bit hatası olup olmadığını kontrol ediyoruz.

Şekil 5.8, Ürt. S için azaltılmış yenileme gecikmesinin tutma hataları üzerindeki etkisini göstermektedir.[4] X-ekseni, darbelemeden önce kurban satırları yenilemek için kullanılan normalize yük yenileme gecikmesini ve y-ekseni, tutma hataları olan satırların sayısını gösterir.

Şekil 5.8'den üç gözlemde bulunuyoruz. Birincisi, Ürt. H ve M'den test edilen satırlar, $0,18t_{RAS(Nom)}$ kullanılarak beş kez yenilendiğinde bile herhangi bir bit hatası yaşamaz. İkincisi, Ürt. S'den test edilen satırlar, $0,27t_{RAS(Nom)}$ kullanılarak yenilendiğinde bit

[4]Ürt. H ve M'den test edilen satırlarda herhangi bir tutma hatası gözlemlemediğimiz için Şekil 5.8'e Ürt. H ve M'yi dahil etmedik.



Şekil 5.8: 2 veri deseni için tekrarlı kısmi yük yenileme uygulandığında veri tutma hatası gözlemleyen satır sayılarının değişimi.

hataları yaşamaya başlar. Üçüncüsü, kısmi yenileme sayısı, bit hatası olan satır sayısını önemli ölçüde etkiler. Örneğin, $0,18t_{RAS(Nom)}$ kullanılarak bir kez yerine beş kez yenilendiğinde, 5657 kat daha fazla satır bit hatası yaşar. Bu gözlemlere dayanarak, Çıkarım 4'ü türetiyoruz.

Çıkarım 4. Yük yenileme gecikmesini (t_{RAS}) bir eşiğe kadar azaltılması, tutma hataları üzerinde etkili olmazken, gecikmeyi bu eşiğin ötesine indirilmesi, tutma hatalarını önemli ölçüde kötüleştirir.

6. PaCRAM

Kısmi Yük Yenileme ile Agresif Önleme (PaCRAM) adlı yeni bir bellek kontrolcüsü tabanlı mekanizma öneriyoruz. PaCRAM'in amacı, mevcut önleyici yenileme tabanlı SatırDarbesi önleme mekanizmalarının performans kaybını azaltmaktır. PaCRAM, bu amacı kısmi yük yenilemeyi kullanarak gerçekleştirir.

6.1 Genel Tasarım

PaCRAM, mevcut bir SatırDarbesi önleme mekanizması ile kullanıldığında, önleyici yük yenileme işlemlerinin gecikmesini dinamik olarak ayarlar. N_{RH} azaltımını telafi etmek için PaCRAM, mevcut SatırDarbesi önleme mekanizmasını azaltılmış N_{RH} ile ayarlar. Bölüm 5.2'de tartışıldığı gibi, PaCRAM, üretici tarafından önerilen zamanlama parametrelerini ihlal ettiği için, tüm yenileme işlemleri için kısmi yük yenilemeyi kullanmak güvenli değildir. Bu nedenle, PaCRAM, bir satırın gerçekleştirebileceği kısmi yük yenilemelerinin sayısını bir eşik (th_{PCR}) ile limitler ve hiçbir satırın nominal gecikme kullanarak tam yenilenmeden, th_{PCR} tane kısmi yük yenileme yapmadığından emin olur. Maksimum kısmi yük yenileme gerçekleştirildikten sonra, PaCRAM, DRAM hücrelerinin yük seviyesini tamamen yenilemek için bir sonraki yenileme işlemi için üretici tarafından önerilen (nominal) gecikmeyi kullanır. Yük seviyesini nominal seviyeye geri getirdikten sonra, PaCRAM yenileme işlemlerinin gecikmesini azaltmaya devam eder. Bunu yaparak, PaCRAM, DRAM çipinin N_{RH} değerinin konfigüre edilen değerin ötesine geçmemesini garanti eder.

6.2 Detaylı Tasarım

Önleyici yenileme işlemlerinin performans kaybını düşük maliyetle azaltmak için, PaCRAM, bir satırın kurbanlarının nominal yenileme gecikmesi kullanılarak tamamen yenilenip yenilenmediğini belirten bir bit vektörünü bellek kontrolcüsünde saklar, buna Tamamen Yenilenmiş (FR) denir. Başlangıçta, PaCRAM, FR'deki tüm bitlere *mantık-1* yazar. SatırDarbesi önleme mekanizması bir önleyici yenileme gerçekleştirdiğinde, önce FR'yi kontrol eder. FR'deki ilgili bit *mantık-1* ise (yani, ilgili satırın kurban satırları tamamen yenilenmemişse), mekanizma önleyici yenileme için nominal yenileme gecikmesini kullanır ve FR'deki ilgili bite *mantık-0* yazar. FR'deki ilgili bit *mantık-0* ise (yani, ilgili satırın kurban satırları tamamen yenilenmişse), mekanizma kısmi yük yenileme için güvenli bir şekilde azaltılmış önleyici yenileme gecikmesini kullanır. Hiçbir satırın th_{PCR} defadan fazla tam olarak yenilenmeden kısmi olarak yenilenmediğini garanti etmek için, PaCRAM periyodik olarak FR'deki tüm bitlere *mantık-1* yazar. PaCRAM, FR'deki tüm bitlerin yenileme periyodunu (t_{FR}) bir satırın th_{PCR} tane ardışık önleyici yenileme alması için gereken minimum süre olarak ayarlar. Örneğin, bir SatırDarbesi önleme mekanizması, bir satır N_{RH}

kez darbelendiğinde bir önleyici yenileme gerçekleştirirse, PaCRAM t_{FR}'yi $th_{PCR} \times ((N_{RH} \times t_{RC}) + t_{refresh})$ olarak ayarlar. Hesaplanan t_{FR} değeri t_{REFW}'den büyükse, PaCRAM, tüm önleyici yenilemeler için azaltılmış yük yenileme gecikmesini kullanır çünkü periyodik yenileme, bir satır th_{PCR} tane kısmi yük yenilemesi almadan önce tam yük yenilemesini gerçekleştirir. Bu şekilde, PaCRAM, bir satırın, SatırDarbesi önleme mekanizmasının th_{PCR} tane kısmi yük yenileme gerçekleştirebileceğinden önce hücrelerin yük seviyesini tamamen yenileyen nominal gecikme ile en az bir kez yenilendiğini garanti eder.

6.3 Donanım Gereksinimleri

PaCRAM'i gerçeklemek için gereken veriler (FR), bellek kontrolcüsünde bir scratchpad SRAM olarak saklanabilir. PaCRAM, her DRAM satırı için bir bit bilgi depolar. CACTI [15] kullanarak PaCRAM'in çip alanı ve erişim gecikmesini analiz ediyoruz. Sonuçlarımız, PaCRAM'in $64K$ DRAM satırından oluşan her DRAM bankası için $0,0069mm^2$'lik bir alan maliyetine sahip olduğunu göstermektedir. Her sırada 16 banka bulunan çift sınıflı bir sisteme entegre edildiğinde, PaCRAM her banka için $8KiB$'lık bir depolama gerektirir ve alan yükü, yüksek kalite bir Intel Xeon işlemcisinin [203, 218] çip/bellek kontrolcüsü alanının $\%0,106/\%1,590$'ına denk gelir. Scratchpad SRAM'ın erişim gecikmesi $0,27ns$ olup, bu gecikme bir satır erişim gecikmesinden küçüktür (örneğin $\sim 14ns$ [186]).

6.4 Güvenlik

PaCRAM, mevcut SatırDarbesi önleme mekanizmalarının güvenlik garantilerinden ödün vermez. PaCRAM, üretici tarafından önerilen yenileme gecikmelerini dikkatlice azaltır ve mekanizmaları buna göre konfigüre eder. Bölüm 6.2'de açıklandığı gibi, PaCRAM, nominal yenileme gecikmesi kullanarak hücrelerin yük seviyesini tamamen yenilemeden hiçbir satırın th_{PCR}'den daha fazla kısmi olarak yenilenmesine izin vermez. Bu şekilde, PaCRAM, herhangi bir satırın N_{RH} değerinin konfigüre edilen N_{RH} değerinden daha düşük olmasına izin vermez. Dolayısıyla, PaCRAM, mevcut SatırDarbesi önleme mekanizmalarının güvenlik garantilerinden ödün vermez.

6.5 Çip-üstünde SatırDarbesi Önleme Mekanizmaları ile PaCRAM

Birçok önceki çalışma [17, 58, 67, 76, 88, 140, 141, 193, 223, 227], SatırDarbesi'ni çip-üstünde önlemeyi önermektedir. Bu çalışmalar, bellek kontrolcüsünün önleyici eylemler yapmasını gerektirmez ve yalnızca DRAM çipi üzerinde çalışır. Örneğin, Panopticon [17], bir DRAM bankasındaki her satırın etkinleştirme sayısını takip eder ve bunları bankanın içinde saklar. Bir satırın sayacı belirli bir eşik değerini aştığında, Panopticon, önleyici yenileme komutuna gerek kalmadan kurban satırlara önleyici yenileme işlemi uygular.

PaCRAM'i çip-üstünde SatırDarbesi önleme mekanizmalarına entegre etmek için, PaCRAM'in DRAM çipinde bulunması ve önleme mekanizması ile iletişim kurması gerekir. Önleme mekanizması, PaCRAM'e önleyici yenileme işlemi ve hedef satır adresleri hakkında bilgi vermelidir. Ardından, PaCRAM, hedef satırlar için azaltılmış yenileme gecikmesinin kullanılıp kullanılamayacağına karar verir ve önleme mekanizmasına bilgi verir. Bu şekilde, PaCRAM, önleyici yenileme işlemleri gerçekleştiren herhangi bir çip-üstünde SatırDarbesi önleme mekanizmasına entegre edilebilir.

6.6 PaCRAM'in Kısıtları

6.6.1 DRAM çiplerinin karakterizasyonu

PaCRAM'in sistem entegrasyonu için, yük yenileme azaltımını doğru bir şekilde tanımlamak gereklidir. Bunu yapmak, kullanılan DRAM çiplerinin profillenmesini gerektirir. Bu profilleme işlemi, metodolojimizde ve önceki çalışmalarda [31, 120, 132] açıklandığı gibi düşük maliyetle farklı şekillerde gerçekleştirilebilir. Birincisi, sistem, DRAM ilk kez başlatıldığında profilleme işlemini başlatabilir ve PaCRAM'i konfigüre eder. İkincisi, DRAM üreticileri, üretim sırasında profilleme işlemini gerçekleştirebilir ve konfigürasyon verilerini Seri Modül Tanıma (SPD) devresine yerleştirebilir [83]. Bellek kontrolcüsü, SPD devresinden konfigürasyon verilerini okuyabilir ve PaCRAM'i konfigüre edebilir. Üçüncüsü, sistem, PaCRAM'i konfigüre etmek için çevrimiçi profilleme gerçekleştirebilir.

6.6.2 Profilleme süresi

Bir DRAM çipinin yük yenileme gecikmesi azaltımını bulmak için birçok metodoloji kullanılabilir. Bir DRAM satırının yük yenileme gecikmesi azaltımını doğru bir şekilde tanımlamak, bir satırı birden fazla yenileme gecikmesi, yenileme sayısı, darbe sayısı ve yinelemeler ile test etmeyi gerektirir. Her test, satırları yazma, yük yenileme, SatırDarbesi saldırısı gerçekleştirme, t_{REFW} sonuna kadar bekleme ve satırı okuma işlemlerinden oluşur. Örneğin, bir metodoloji, bir satırın yenileme gecikmesi azaltımını doğru bir şekilde belirlemek için her satırı 5 yenileme gecikmesi, 10 yenileme sayısı, 5 darbe sayısı ve 5 yineleme için test edebilir. Profilleme maliyetini nicel olarak göstermek için, $64K$ satıra sahip bir DDR4 DRAM bankası için yenileme gecikmesi azaltımını belirlemek amacıyla bir profilleme metodolojisi örneği sunuyoruz.

Profilleme metodolojisi, her satırın yenileme gecikmesi azaltımını tanımlamayı gerektirir. Bankanın gecikme azaltımını bulmak için, bankadaki tüm satırlar arasında minimum yenileme gecikmesi azaltımını alır. Bir satırın test süresi büyük ölçüde t_{REFW} sonuna kadar uzun bekleme süresi tarafından domine edildiği için, birden fazla satırın testlerini iç içe geçirebiliriz (yani, bir satırı yazdıktan ve saldırgan satırlarını darbeledikten sonra, ilk satır t_{REFW} sonuna kadar beklerken başka bir

satırı test etmeye başlayabiliriz). Bunu yaparak, bir satıra yazmak ve saldırganlarını darbelemek yaklaşık olarak $\sim 0,05ms$ sürdüğünden, ilk satırı beklerken ~ 1270 satırı test etmeye başlayabiliriz [88]. Bu nedenle, bir DRAM bankasında ~ 1270 satırı test etmek yaklaşık $128ms$ sürer (yani, ilk satır t_{REFW}'ye ulaşmadan hemen önce son satırı test etmeye başlarız). $64K$ satırlı bir DRAM bankasındaki tüm satırları test etmek $\sim 6,7s$ $(128ms \times (64K/1270))$ sürer. $64K$ satırlı bir bankanın yük yenileme gecikmesi azaltımını doğru bir şekilde belirlemek için, profilleme metodolojisi $\sim 2,3$ saat $(6,7s \times 5 \times 10 \times 5 \times 5)$ sürer. Profilleme metodolojisinin ileri optimizasyonlarını gelecekteki çalışmalara bırakıyoruz.

6.6.3 Sistem güvenilirliği

PaCRAM, t_{RAS} zamanlama kısıtlamasında koruma bantları uygulayan DRAM çiplerini kullanan sistemlerle sınırlıdır. Modern gerçek DDR4 DRAM çiplerinde t_{RAS} zamanlama kısıtlamasında önemli ölçüde büyük koruma bantlarının bulunduğunu deneysel olarak gösteriyoruz. Bulgularımız, ticari DRAM çiplerinde birkaç zamanlama kısıtlamasının (örneğin, t_{RCD}, t_{WR}, t_{RP}, t_{RAS}) büyük koruma bantları uyguladığını gösteren birçok önceki çalışmayla [29, 33, 107, 120, 121, 132, 142, 224] uyumludur. Bu tür koruma bantlarının olmadığı durumlarda, PaCRAM hata düzeltme mekanizmaları [20, 35, 47, 65, 74, 102, 111, 150, 234] ile birleştirilebilir. Bu tür sistemlerin analizini gelecekteki çalışmalara bırakıyoruz.

6.6.4 Az sayıda/hiç önleyici yenileme olmadan PaCRAM

PaCRAM, önleyici yenilemelere harcanan toplam süreyi azaltarak SatırDarbesi önleme mekanizmalarının performans kaybını azaltır. PaCRAM'in faydaları, büyük ölçüde önleyici yenileme sayısına bağlıdır. Bu nedenle, bir mekanizma daha az önleyici yenileme ile SatırDarbesi'ni önlemeyi başarırsa, PaCRAM'in faydaları azalır. Önceki çalışmalar [47, 173, 185, 190, 219, 226], önleyici yenilemeler dışında önleyici eylemler (örneğin, sınırlama [226], satır değiştirme [185, 190, 219], ve ECC [47, 173]) gerçekleştiren SatırDarbesi önleme mekanizmaları da önermektedir. PaCRAM, önleyici yenilemelerin yük yenileme gecikmesini azalttığı için, PaCRAM bu çalışmalarla birlikte kullanılamaz.

7. PaCRAM'İN DEĞERLENDİRİLMESİ

Bu bölümde, yük yenileme gecikmesini azaltmanın potansiyel faydalarını, PaCRAM'i beş son teknoloji SatırDarbesi önleme mekanizması [89, 91, 114, 166, 174] ile kullanarak gösteriyoruz.

7.1 Analiz Metodolojisi

Simülasyon ortamı. PaCRAM'in sistem performansına etkisini değerlendirmek için, Ramulator 2.0 [60, 138] kullanarak simülasyonlar gerçekleştiriyoruz. Çizelge 7.1, simüle edilen sistem konfigürasyonunu göstermektedir. Değerlendirmelerimizde, DDR5 DRAM modülüne bağlı tek/dört çekirdekli gerçekçi bir sistem kullanıyoruz. Bellek kontrolcüsü, dört limitli [202] FR-FCFS [179, 245] sıralama politikasını kullanmaktadır.

Çizelge 7.1: Simüle edilen sistem konfigürasyonu.

İşlemci	1/4 çekirdekli, 3,2GHz saat frekansı, 4-genişlikli getirme, 128-girdili buyruk penceresi
DRAM	DDR5, 1 kanal, 2 sınıf, 8 banka grubu, 2 banka/banka grubu, 65K satır/banka
Bellek Kontrolcüsü	64-girdili okuma ve yazma istek kuyruğu, Sıralama politikası: FR-FCFS [179, 245] Adres haritası: MOP[98]
Son-seviye Önbellek	Çekirdek başına 2 MiB

Karşılaştırma noktaları. Temel sistemimiz hiçbir SatırDarbesi önleme mekanizması kullanmamaktadır. PaCRAM'i, beş önleyici yenileme tabanlı SatırDarbesi önleme mekanizması, PARA [114], RFM [89], PRAC [91], Hydra [174] ve Graphene [166] ile değerlendiriyoruz.

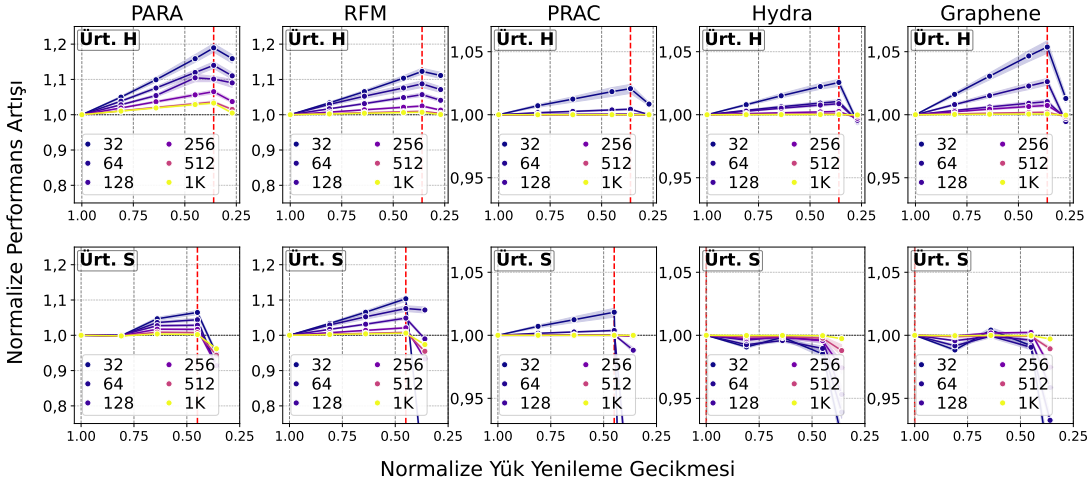
Test programları. PaCRAM'i beş kıyaslama seti kullanarak değerlendiriyoruz: SPEC CPU2006 [199], SPEC CPU2017 [200], TPC [209], MediaBench [53] ve YCSB [39]. Rastgele olarak 62 tek çekirdekli test programı ve her biri 4 test programından oluşan 60 çok programlı test programı karışımı seçiyoruz. Her test programı için SimPoint [196] kullanarak $100M$ buyruğa karşılık gelen bellek istek kaydını oluşturuyoruz. Önceki çalışmalara benzer şekilde [110, 225, 226], bu istekleri, her çekirdeğin $10M$ buyruktan oluşan bir ısınma süresi ile $100M$ buyruk yürütmesine kadar simüle ediyoruz.

Ölçütler. PaCRAM'in sistem performansı (çevrim başına buyruk (ÇBB) ve ağırlıklı hızlanma (AH)), önleyici yenilemelere harcanan toplam süre ve DRAM modülünün enerji tüketimi üzerindeki etkisini değerlendiriyoruz.

PaCRAM'in konfigüre edilmesi. PaCRAM'in performans etkisini göstermek için, PaCRAM'i karakterizasyon sonuçlarımızı kullanarak konfigüre ediyoruz. t_{RAS} koruma bantlarının sırasıyla büyük ve küçük olduğu iki modül seçiyoruz; biri Ürt. H'den ve diğeri Ürt. S'den. Her modül için, tüm çipler arasında en kötü durumu temsil eden N_{RH} (yani, en küçük N_{RH}) değerini kullanıyoruz. PaCRAM-H/S, Ürt. H/S'den gelen modülle konfigüre edilen versiyonu temsil eder. PaCRAM-H/S, dört/üç farklı yük yenileme gecikmesi değeri, N_{RH} azaltma oranı içerir. Bölüm 6.2'de açıklandığı gibi, her SatırDarbesi önleme mekanizması ve PaCRAM konfigürasyonu için gerekli parametreleri ayrı ayrı hesaplıyoruz.

7.2 Gecikme Azaltmasına Hassasiyet

Gecikme azaltma miktarının PaCRAM'in performansını nasıl etkilediğini anlamak için, Ürt. H ve Ürt. S için çeşitli gecikme azaltma değerleri kullanıyoruz. Şekil 7.1, gecikme azaltma miktarının beş SatırDarbesi savunması ve iki üretici üzerindeki performansını nasıl etkilediğini göstermektedir. X-ekseni, normalize yenileme gecikmesini, y-ekseni ise önleyici yenilemelerin gerçekleştirilmesinde nominal gecikmeler kullanıldığında test programlarının ÇBB dağılımına göre normalize edilmiş ÇBB değerlerini göstermektedir. Her eğri, farklı N_{RH} değeri temsil eder.



Şekil 7.1: Farklı gecikme miktarları için PaCRAM'in performans artışı.

Şekil 7.1'den üç gözlemde bulunuyoruz. Birincisi, Ürt. H için azaltılmış yük yenileme gecikmesi kullanmak, her azaltılmış gecikme ve N_{RH} değerinde test edilen her SatırDarbesi önleme mekanizmasının performansını artırır. İkincisi, PaCRAM'in performansı, bir azaltma eşiğine kadar artar, ardından eşiğin ötesinde azalır çünkü yük yenileme gecikmesinin azaltılması SatırDarbesi zafiyetini artırır (yani, DRAM çipinin N_{RH} değeri düşer). Üçüncüsü, Ürt. S için, yenileme gecikmesinin azaltılması Hydra ve Graphene'nin performansını azaltır çünkü yenileme gecikmesinin azaltılması SatırDarbesi zafiyetini artırır ve daha fazla yenileme işlemi gerektirir. Ürt. H için optimal yük yenileme gecikmesi $0,36t_{RAS}$ iken, Ürt. S için RFM ve PARA için optimal yenileme gecikmesi $0,45t_{RAS}$'dir (kırmızı kesik çizgilerle gösterilmiştir). Ürt. H'den

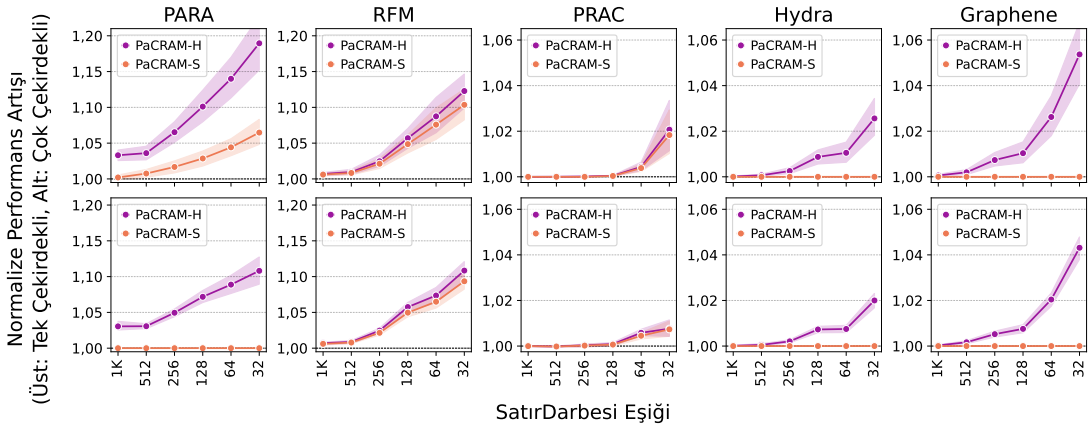
gelen DRAM çipleri, Ürt. S'den gelen çiplere göre daha büyük koruma bantlarına sahip olduğundan ve azaltılmış yenileme gecikmelerine karşı daha dayanıklı olduğundan, PaCRAM, Ürt. H için yenileme gecikmesini Ürt. S'den daha fazla azaltabilir. Bu gözlemlerden Çıkarım 5'i türetiyoruz.

Çıkarım 5. Sistem performansı, yük yenileme gecikmesini azalttıkça önce artar, sonra azalır.

Geri kalan performans analizimiz için PaCRAM, Ürt. H/S için optimal yük yenileme gecikmesini kullanır.

7.3 PaCRAM'in Performans Üzerine Etkisi

Şekil 7.2, altı farklı N_{RH} değeri (x-ekseni) ve beş farklı SatırDarbesi önleme mekanizması (sütunlar) için PaCRAM'in, temel SatırDarbesi önleme mekanizmalarına göre normalize edilmiş performans iyileştirmesini (y-ekseni) göstermektedir. Üst/alt grafikler, tek çekirdekli/çok çekirdekli sistem için ÇBB/AH türünden performansı temsil eder. Her eğri, farklı bir PaCRAM konfigürasyonunu temsil eder. Eğrilerin etrafındaki gölgeli alanlar, %95 güven aralıklarını gösterir.



Şekil 7.2: PaCRAM'in performans artışı.

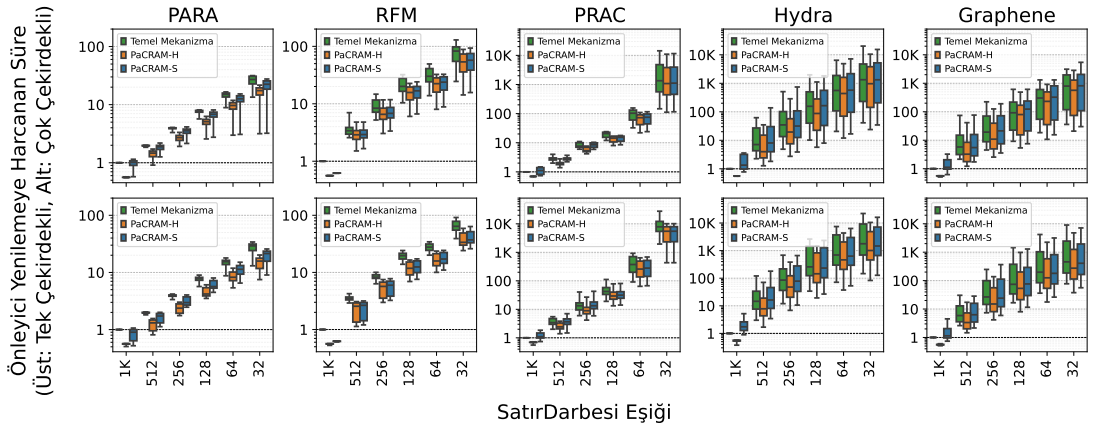
Şekil 7.2'den beş gözlemde bulunuyoruz. Birincisi, PaCRAM-H, test edilen tüm SatırDarbesi önleme mekanizmalarının performans kaybını azaltarak tüm N_{RH} değerleri ve konfigürasyonları için sistem performansını artırır. Örneğin, PaCRAM-H, N_{RH} değeri 32 olan PARA/RFM/PRAC/Hydra/Graphene için sırasıyla ortalama %18,95/%12,28/%2,07/%2,56/%5,37 oranında sistem performansını artırır. İkincisi, tüm SatırDarbesi önleme mekanizmaları için N_{RH} azaldıkça PaCRAM-H'nin performans iyileştirmesi artar. Örneğin, PaCRAM-H, N_{RH} değeri $1K/32$ olan RFM için ortalama $\%0,69/\%12,28$ oranında sistem performansını artırır. Üçüncüsü, PaCRAM-S'nin performans iyileştirmesi, PaCRAM-H'nin performans iyileştirmesinden daha düşüktür. PaCRAM-H, ortalama sistem performansını PaCRAM-S'ye kıyasla $2,21$ kat daha fazla artırır. Dördüncüsü, PaCRAM, PRAC, Hydra ve Graphene'ya kıyasla PARA ve RFM üzerinde daha büyük performans

iyileştirmeleri sağlar. PRAC, Hydra ve Graphene, PARA ve RFM'ye kıyasla daha az SatırDarbesi önleyici yenileme gerçekleştirir ve daha yüksek donanım karmaşıklığı pahasına daha küçük performans kayıplarına neden olur, bu nedenle PaCRAM, PARA veya RFM ile kullanıldığında daha fazla önleyici yenileme gecikmesini azaltabilir. Beşinci olarak, PaCRAM'in tek çekirdekli sistem için sağladığı iyileştirmeler, çok çekirdekli sistemden biraz daha yüksektir. Tek çekirdekli sistem, PaCRAM'den çok çekirdekli sisteme kıyasla $1,43$ kat daha fazla faydalanır. Bu gözlemlerden Çıkarım 6'yı türetiyoruz.

Çıkarım 6. PaCRAM, hem tek çekirdekli hem de çok çekirdekli sistemler için sistem performansını önemli ölçüde artırır.

7.4 PaCRAM'in Önleyici Yenileme Süreleri Üzerine Etkisi

PaCRAM'in performans iyileştirmesinin arkasındaki nedeni anlamak için, önleyici yenilemelere harcanan toplam süreyi analiz ediyoruz. Şekil 7.3, tek çekirdekli (üst grafikler) ve çok çekirdekli (alt grafikler) sistemler için önleyici yenilemelere harcanan toplam süreyi göstermektedir. X-ekseni altı farklı N_{RH} değerini ve y-ekseni $N_{RH} = 1K$ olan temel SatırDarbesi önleme mekanizmalarına normalize edilmiş önleyici yenilemelere harcanan toplam süreleri göstermektedir.



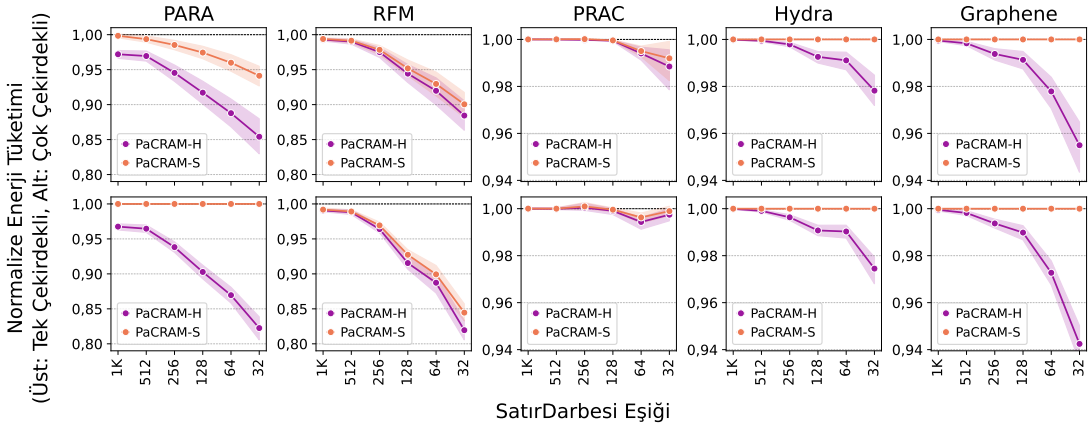
Şekil 7.3: PaCRAM'in önleyici yük yenilemelerine harcadığı toplam sürenin dağılımı.

Şekil 7.3'ten üç gözlemde bulunuyoruz. Birincisi, yenilemelere harcanan zaman N_{RH} azaldıkça önemli ölçüde artar. Örneğin, N_{RH} $1K$'dan 32'e düşürüldüğünde önleyici yenilemelere harcanan zaman 144 kat artar. İkincisi, PaCRAM, tüm konfigürasyonlarda RFM ve PARA için toplam yenileme süresini önemli ölçüde azaltır. PaCRAM-H/-S, tüm N_{RH} değerleri boyunca ortalama olarak önleyici yenilemelere harcanan toplam süreyi sırasıyla $\%34,12/\%10,70$ oranında azaltır. Üçüncüsü, PaCRAM-S, PaCRAM-H'den daha fazla yenileme süresi harcar. Bu gözlemlerden Çıkarım 7'yi türetiyoruz.

Çıkarım 7. PaCRAM, önleyici yenilemelere harcanan toplam süreyi önemli ölçüde azaltır.

7.5 PaCRAM'in Enerji Kullanımı Üzerine Etkisi

Sonraki adımda, DRAM modülünün enerji tüketimini analiz ediyoruz. Bunu yapmak için, Ramulator 2.0 [60, 138] simülatörünü DRAMPower [30] altyapısı ile genişletiyoruz. Şekil 7.4, altı farklı N_{RH} değeri (x-ekseni) için PaCRAM'in enerji tüketimini (y-ekseni) temel SatırDarbesi savunmalarının tüketimine normalize edilmiş şekilde göstermektedir. İki eğri, PaCRAMiçin farklı konfigürasyonlarını göstermektedir.



Şekil 7.4: PaCRAM'in enerji tüketimi.

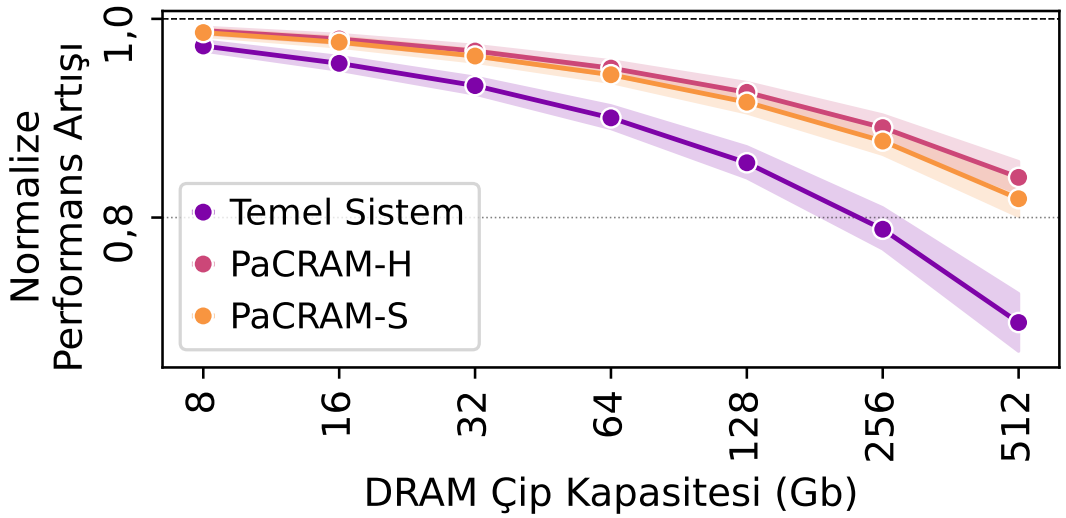
Şekil 7.4'ten iki gözlemde bulunuyoruz. Birincisi, PaCRAM-H, tüm önleme mekanizmaları ve N_{RH} değerleri için enerji tüketimini azaltır. PaCRAM-H, N_{RH} değeri 32 olduğunda PARA/RFM/PRAC/Hydra/Graphene'nin enerji tüketimini ortalama %14,59/%11,56/%1,15/%2,18/%4,50 oranında azaltır. İkincisi, PaCRAM-S, PaCRAM-H'den daha fazla enerji tüketir. Bu gözlemlerden Çıkarım 8'i türetiyoruz.

Çıkarım 8. PaCRAM, DRAM'in enerji tüketimini önemli ölçüde azaltır.

7.6 PaCRAM'in Potansiyel Eklentileri

PaCRAM, periyodik yük yenileme işlemleri ve dinamik erişimler için yük yenileme gecikmesini azaltmak amacıyla genişletilebilir. Bu tür eklentilerin potansiyel faydalarını göstermek için, PaCRAM'i periyodik yük yenilemelerini azaltılmış gecikme ile gerçekleştirecek şekilde güncelliyoruz. Şekil 7.5, farklı DRAM çip yoğunlukları (x-ekseni) için hiçbir yenileme işlemi gerçekleştirmeyen ideal bir sisteme normalize edilmiş şekilde sistem performansını ÇBB türünde (y-ekseni) göstermektedir. Üç eğri, nominal yenileme gecikmesi olan temel sistemi ve Ürt. H ve Ürt. S için azaltılmış periyodik yenileme gecikmesi ile PaCRAM'i temsil etmektedir.

Şekil 7.5'ten iki gözlemde bulunuyoruz. Birincisi, PaCRAM, tüm konfigürasyonlar ve DRAM çip yoğunlukları için sistem performansını önemli ölçüde artırır. Örneğin, PaCRAM-H, 512*Gib* DRAM çipinde sistem performansını %22,37 oranında artırır. İkincisi, periyodik yenileme yükü, yenileme gecikmesinin artması nedeniyle DRAM



Şekil 7.5: Azaltılmış periyodik yük yenileme gecikmesi uygulandığında performans artışı.

çip yoğunluğu arttıkça artar. Bu gözlemlerden yola çıkarak, periyodik yenileme işlemlerinin gecikmesini azaltmanın, PaCRAM'in performans faydalarını artırdığı sonucuna varıyoruz. Bu genişletme, PaCRAM'in veri yönetiminde değişiklikler gerektirir. Bu eklentiyi gelecekteki çalışmalara bırakıyoruz.

8. İLGİLİ ÇALIŞMALAR

Bildiklerimize göre, bu çalışma, yük yenileme gecikmesinin azaltılmasının gerçek bir DRAM çipinin SatırDarbesi zafiyetini nasıl etkilediğini ve bunun son teknoloji SatırDarbesi önleme mekanizmaları üzerindeki etkilerini deneysel olarak inceleyen ilk çalışmadır. Önleyici yenileme işlemlerine harcanan zamanı azaltarak mevcut SatırDarbesi savunmalarının performans kayıplarını azaltan düşük maliyetli bir mekanizma olan PaCRAM'i öneriyoruz. Bölüm 7, PaCRAM'in faydalarını nicel olarak göstermektedir. Bu bölümde, ilgili geçmiş çalışmaları üç kategoride açıklıyoruz: 1) azaltılmış gecikme altında DRAM işleminin araştırmaları, 2) gerçek DRAM çiplerinin SatırDarbesi zafiyetinin karakterizasyon çalışmaları ve 3) SatırDarbesi saldırıları ve savunmaları.

8.1 Azaltılmış Gecikme Altında DRAM İşlemi

Birçok çalışma, azaltılmış gecikme altında DRAM'i karakterize eder [10, 29, 34, 54, 55, 107–109, 120, 121, 157, 160, 225] ve üretici tarafından önerilen DRAM işlem gecikmelerini ihlal ederek sistem performansını ve enerji verimliliğini artıran teknikler geliştirir [29, 34, 40, 67, 69, 107, 118, 120–122, 137, 158, 172, 197, 215, 225, 235], rastgele sayılar veya fiziksel olarak kopyalanamaz fonksiyonlar üretmek için DRAM çiplerini kullanır [10, 22, 108, 109, 155, 157, 160] ve bellek kullanarak işlem yapma tekniklerini uygular [54, 55]. Ancak, bu çalışmaların hiçbiri, azaltılmış yük yenileme gecikmesinin SatırDarbesi üzerindeki etkisini incelemez, bu da SatırDarbesi saldırısı altında güvenlik sağlamayan çözümlerle sonuçlanabilir. Biz ise bu konuyu ele alır ve topladığımız gözlemleri kullanarak sistem performansını ve enerji verimliliğini artırırız. Hem güvenliği hem de güvenilirliği sağlamak için bir mekanizma, yenilikçi gözlemlerimizden faydalanabilir. Çalışmamız PaCRAM'i tanıtırken, gözlemlerimizi önceki çalışmalara entegre etme analizini gelecekteki araştırmalara bırakıyoruz.

8.2 SatırDarbesi Zafiyetinin Deneysel Karakterizasyonu

Önceki çalışmalar, gerçek DRAM çiplerinde SatırDarbesi zafiyetini kapsamlı bir şekilde karakterize eder [52, 70, 110, 114, 136, 161, 164]. Bu çalışmalar, gerçek DDR3, DDR4 ve LPDDR4 DRAM çiplerini kullanarak bir DRAM çipinin SatırDarbesi zafiyetinin 1) DRAM yenileme penceresi [52, 70, 114], 2) saldırgan ve kurban satırlar arasındaki fiziksel mesafe [110, 114], 3) DRAM nesli ve teknoloji düğümü [70, 110, 114, 161], 4) sıcaklık [161, 164], 5) saldırgan satırın aktif kaldığı süre [136, 161, 164], 6) kurban DRAM hücresinin fiziksel konumu [161] ve 7) azaltılmış kelime hattı voltajı [224] ile nasıl değiştiğini gösterir. Ancak, bu çalışmaların hiçbiri, yük yenileme gecikmesinin gerçek DRAM çiplerinde SatırDarbesi zafiyeti üzerindeki etkisini araştırmaz. Karakterizasyon çalışmamız, bu analizleri geliştirerek

SatırDarbesi'nin davranışı hakkında yeni gözlemler yapar.

8.3 SatırDarbesi Saldırıları ve Önleme Mekanizmaları

Birçok önceki çalışma [1, 8, 16, 18, 19, 21, 23–25, 28, 36–38, 45, 46, 48, 51, 52, 61, 62, 70, 75, 79, 81, 82, 92, 99, 110, 114, 116, 119, 125, 127, 131, 133, 140, 145, 146, 148, 159, 161, 164, 165, 169–171, 173, 175–178, 180, 182, 188, 191, 192, 201, 204–207, 210, 211, 213, 214, 216, 221, 229–231, 236, 238, 240, 241], SatırDarbesi'nin sistem güvenliğini ve emniyetini tehlikeye atmak için sistem düzeyinde saldırılar düzenlemek amacıyla kullanılabileceğini göstermektedir (örneğin, kök ayrıcalıklarını elde etmek veya özel verileri sızdırmak). Bu saldırılara karşı koruma sağlamak için, birçok önceki çalışma [3–7, 11–14, 17, 23, 27, 42–44, 47, 49, 50, 52, 56, 58, 59, 63, 64, 66–68, 70, 73, 76, 78, 88, 94–97, 106, 112–114, 117, 123, 124, 126, 134, 135, 139–141, 149, 151, 152, 162, 166, 173, 174, 181, 185, 187–190, 193, 195, 198, 208, 211, 212, 217, 220, 223, 225–228, 232, 237, 239, 243, 244], bit hatalarının bir sistemi tehlikeye atmasını önleyen SatırDarbesi önleme mekanizmalarını önermektedir. Bölüm 7'da gösterdiğimiz gibi, bu çalışmada yaptığımız yenilikçi gözlemler önleyici yenilemelere harcanan zamanı azaltarak mevcut SatırDarbesi önleme mekanizmalarının performans kaybını azaltmak için kullanılabilir.

9. SONUÇ

Bu çalışmada, üç büyük DRAM üreticisinden 388 DDR4 DRAM çipinde yük yenileme, SatırDarbesi ve veri tutma hataları arasındaki etkileşimler üzerine yapılan ilk kapsamlı deneysel çalışmayı sunuyoruz. Deneylerimizde, potansiyel kurban satırlarının, hafifçe daha fazla ($\%0,54$) önleyici yenileme işlemi gerektirme pahasına, önemli ölçüde daha düşük ($\%64$) gecikme ile önleyici olarak yenilenebileceğini gözlemledik. Bu gözlemler, yük yenileme gecikmesinin azaltılmasının DRAM çiplerinin SatırDarbesi zafiyeti üzerindeki etkilerini anlamak için kritik öneme sahiptir.

Bu gözlemleri kullanarak, mevcut SatırDarbesi önleme mekanizmalarının yük yenileme gecikmesini azaltan ve sistem performansını artıran, bellek kontrolcüsü tabanlı düşük maliyetli bir mekanizma olan Kısmi Yük Yenileme ile Agresif Önleme (PaCRAM) adlı bir mekanizma öneriyoruz. PaCRAM, mevcut SatırDarbesi önleme mekanizmalarının performans kaybını azaltmak amacıyla geliştirilen, önleyici yenileme işlemlerinin gecikmesini dinamik olarak ayarlayan bir sistemdir. Bu mekanizma, beş farklı SatırDarbesi önleme mekanizmasının neden olduğu performans (enerji) kayıplarını sırasıyla ortalama olarak %18,95 (%14,59), %12,28 (%11,56), %2,07 (%1,15), %2,56 (%2,18) ve %5,37 (%4,50) oranında azaltarak sistem performansını ve enerji verimliliğini önemli ölçüde artırdığını gösteriyoruz.

PaCRAM'in geliştirilmesi, DRAM tabanlı bellek sistemlerinde daha yüksek verimlilik ve güvenilirlik sağlamak için önemli bir adım teşkil etmektedir. Yaptığımız çalışmada, yük yenileme gecikmesinin azaltılmasının DRAM çiplerinin SatırDarbesi zafiyeti üzerindeki etkilerini detaylı bir şekilde inceledik ve bu incelemelerden elde edilen yeni çıkarımlarda bulunduk. Simülasyon sonuçlarımız, PaCRAM'in düşük maliyetli ve etkili bir çözüm olduğunu göstermektedir. PaCRAM, DRAM çiplerinde daha yüksek verimlilik ve güvenilirlik sağlarken, aynı zamanda performans kayıplarını da minimize etmektedir.

PaCRAM'in uygulanabilirliği ve etkileri üzerine yapılan bu çalışma, gelecekteki DRAM tabanlı bellek sistemlerinin tasarımı ve geliştirilmesi için önemli katkılar sunmaktadır. PaCRAM, mevcut SatırDarbesi önleme mekanizmaları ile uyumlu bir şekilde çalışarak, sistem güvenliğini ve performansını artırmak için yenilikçi bir yaklaşım sağlamaktadır. Bu çalışma, yük yenileme gecikmesi azaltımının, DRAM çiplerinin güvenilirliğini ve performansını nasıl etkilediğine dair değerli bilgiler sunmakta ve gelecekteki araştırmalar için bir temel oluşturmaktadır.

KAYNAKLAR

[1] **Aga**, **M. T.**, **Aweke**, **Z. B.**, and **Austin**, **T.** (2017). When Good Protections Go Bad: Exploiting Anti-DoS Measures to Accelerate Rowhammer Attacks. In: *HOST*.

[2] **Agarwal**, **S.**, **Dixit**, **H.**, **Datta**, **D.**, **Tran**, **M.**, **Houssameddine**, **D.**, **Shum**, **D.**, and **Benistant**, **F.** (2018). Rowhammer for Spin Torque based Memory: Problem or not? In: *INTERMAG*.

[3] **Aichinger**, **B.** (2015). DDR Memory Errors Caused by Row Hammer. In: *HPEC*.

[4] **Ajorpaz**, **S. M.**, **Moghimi**, **D.**, **Collins**, **J. N.**, **Pokam**, **G.**, **Abu-Ghazaleh**, **N.**, and **Tullsen**, **D.** (2022). EVAX: Towards a Practical, Pro-active & Adaptive Architecture for High Performance & Security. In: *MICRO*.

[5] **Apple Inc.** (2015). About the Security Content of Mac EFI Security Update 2015-001. `https://support.apple.com/en-us/HT204934`.

[6] **Arıkan**, **K.**, **Palumbo**, **A.**, **Cassano**, **L.**, **Reviriego**, **P.**, **Pontarelli**, **S.**, **Bianchi**, **G.**, **Ergin**, **O.**, and **Ottavi**, **M.** (2022). Processor security: Detecting microarchitectural attacks via count-min sketches. In: *VLSI*.

[7] **Aweke**, **Z. B.**, **Yitbarek**, **S. F.**, **Qiao**, **R.**, **Das**, **R.**, **Hicks**, **M.**, **Oren**, **Y.**, and **Austin**, **T.** (2016). ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks. In: *ASPLOS*.

[8] **Aydin**, **H.** and **Sertbaş**, **A.** (2022). Cyber Security in Industrial Control Systems (ICS): A Survey of RowHammer Vulnerability. In: *Applied Computer Science*.

[9] **Baeg**, **S.**, **Yun**, **D.**, **Chun**, **M.**, and **Wen**, **S.-J.** (2022). Estimation of the Trap Energy Characteristics of Row Hammer-Affected Cells in Gamma-Irradiated DDR4 DRAM. In: *IEEE Transactions on Nuclear Science*.

[10] **Bahar Talukder**, **B. M. S.**, **Ray**, **B.**, **Forte**, **D.**, and **Rahman**, **M. T.** (2019). PreLatPUF: Exploiting DRAM Latency Variations for Generating Robust Device Signatures. In: *IEEE Access*.

[11] **Bains**, **K.** et al. (2015). Row Hammer Refresh Command. US Patents: 9,117,544 9,236,110 10,210,925.

[12] **Bains**, **K.**, **Halbert**, **J.**, **Mozak**, **C.**, **Schoenborn**, **T.**, and **Greenfield**, **Z.** (2015). Row Hammer Refresh Command.

[13] **Bains**, **K. S.** and **Halbert**, **J. B.** (2016a). Distributed Row Hammer Tracking.

[14] — (2016b). Row Hammer Monitoring Based on Stored Row Hammer Threshold Value. US Patent: 10,083,737.

[15] **Balasubramonian**, **R.**, **Kahng**, **A. B.**, **Muralimanohar**, **N.**, **Shafiee**, **A.**, and **Srinivas**, **V.** (2017). CACTI: An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model. `https://www.hpl.hp.com/research/cacti/`.

[16] **Barenghi**, **A.**, **Breveglieri**, **L.**, **Izzo**, **N.**, and **Pelosi**, **G.** (2018). Software-Only Reverse Engineering of Physical DRAM Mappings for Rowhammer Attacks. In: *IVSW*.

[17] **Bennett**, **T.**, **Saroiu**, **S.**, **Wolman**, **A.**, and **Cojocar**, **L.** (2021). Panopticon: A Complete In-DRAM Rowhammer Mitigation. In: *DRAMSec*.

[18] **Bhattacharya**, **S.** and **Mukhopadhyay**, **D.** (2016). Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis. In: *CHES*.

[19] — (2018). Advanced Fault Attacks in Software: Exploiting the Rowhammer Bug. In: *Fault Tolerant Architectures for Cryptography and Hardware Security*. Springer Singapore.

[20] **Bose**, **R. C.** and **Ray-Chaudhuri**, **D. K.** (1960). On a Class of Error Correcting Binary Group Codes. In: *Information and control*.

[21] **Bosman**, **E.**, **Razavi**, **K.**, **Bos**, **H.**, and **Giuffrida**, **C.** (2016). Dedup Est Machina: Memory Deduplication as An Advanced Exploitation Vector. In: *S&P*.

[22] **Bostanci**, **F. N.**, **Olgun**, **A.**, **Orosa**, **L.**, **Yaglikci**, **A. G.**, **Kim**, **J. S.**, **Hassan**, **H.**, **Ergin**, **O.**, and **Mutlu**, **O.** (2022). DR-STRaNGe: End-to-End System Design for DRAM-based True Random Number Generators. In: *HPCA*.

[23] **Brasser**, **F.**, **Davi**, **L.**, **Gens**, **D.**, **Liebchen**, **C.**, and **Sadeghi**, **A.-R.** (2017). Can't Touch This: Software-Only Mitigation Against Rowhammer Attacks Targeting Kernel Memory. In: *USENIX Security*.

[24] **Burleson**, **W.**, **Mutlu**, **O.**, and **Tiwari**, **M.** (2016). Invited: Who is the Major Threat to Tomorrow's Security? You, the Hardware Designer. In: *DAC*.

[25] **Cai**, **K.**, **Zhang**, **Z.**, and **Yao**, **F.** (2022). On the Feasibility of Training-time Trojan Attacks through Hardware-based Faults in Memory. In: *HOST*.

[26] **Canpolat**, **O.**, **Yaglikçi**, **A. G.**, **Oliveira**, **G.**, **Olgun**, **A.**, **Ergin**, **O.**, and **Mutlu**, **O.** (2024). Understanding the Security Benefits and Overheads of Emerging Industry Solutions to DRAM Read Disturbance. In: *DRAMSec*.

[27] **Canpolat**, **O.**, **Yağlıkçı**, **A. G.**, **Olgun**, **A.**, **Emir Yüksel**, **İsmail**, **Tuğrul**, **Y. C.**, **Kanellopoulos**, **K.**, **Ergin**, **O.**, and **Mutlu**, **O.** (2024). Leveraging Adversarial Detection to Enable Scalable and Low Overhead RowHammer Mitigations. arXiv: `2404.13477 [cs.CR]`.

[28] **Carre**, **S.**, **Desjardins**, **M.**, **Facon**, **A.**, and **Guilley**, **S.** (2018). OpenSSL Bellcore's Protection Helps Fault Attack. In: *DSD*.

[29] **Chandrasekar**, **K.**, **Goossens**, **S.**, **Weis**, **C.**, **Koedam**, **M.**, **Akesson**, **B.**, **Wehn**, **N.**, and **Goossens**, **K.** (2014). Exploiting Expendable Process-Margins in DRAMs for Run-Time Performance Optimization. In: *DATE*.

[30] **Chandrasekar**, **K.**, **Weis**, **C.**, **Li**, **Y.**, **Goossens**, **S.**, **Jung**, **M.**, **Naji**, **O.**, **Akesson**, **B.**, **Wehn**, **N.**, and **Goossens**, **K.** (n.d.). DRAMPower: Open-Source DRAM Power & Energy Estimation Tool. `http://www.drampower.info/`.

[31] **Chang**, **K.** et al. (2017). Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms. In: *SIGMETRICS*.

[32] **Chang**, **K. K.** (2017). Understanding and Improving the Latency of DRAM-Based Memory Systems. PhD thesis. Carnegie Mellon University.

[33] **Chang**, **K. K.**, **Kashyap**, **A.**, **Hassan**, **H.**, **Ghose**, **S.**, **Hsieh**, **K.**, **Lee**, **D.**, **Li**, **T.**, **Pekhimenko**, **G.**, **Khan**, **S.**, and **Mutlu**, **O.** (2016). Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization. In: *SIGMETRICS*.

[34] **Chang**, **K. K.**, **Yağlıkçı**, **A. G.**, **Ghose**, **S.**, **Agrawal**, **A.**, **Chatterjee**, **N.**, **Kashyap**, **A.**, **Lee**, **D.**, **O'Connor**, **M.**, **Hassan**, **H.**, and **Mutlu**, **O.** (2017). Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms. In: *SIGMETRICS*.

[35] **Chen**, **L.** and **Zhang**, **Z.** (2014). MemGuard: A low cost and energy efficient design to support and enhance memory system reliability. In: *ISCA*.

[36] **Cohen**, **Y.**, **Tharayil**, **K. S.**, **Haenel**, **A.**, **Genkin**, **D.**, **Keromytis**, **A. D.**, **Oren**, **Y.**, and **Yarom**, **Y.** (2022). HammerScope: Observing DRAM Power Consumption Using Rowhammer. In: *CCS*.

[37] **Cojocar**, **L.**, **Kim**, **J.**, **Patel**, **M.**, **Tsai**, **L.**, **Saroiu**, **S.**, **Wolman**, **A.**, and **Mutlu**, **O.** (2020). Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers. In: *S&P*.

[38] **Cojocar**, **L.**, **Razavi**, **K.**, **Giuffrida**, **C.**, and **Bos**, **H.** (2019). Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In: *S&P*.

[39] **Cooper**, **B.**, **Silberstein**, **A.**, **Tam**, **E.**, **Ramakrishnan**, **R.**, and **Sears**, **R.** (2010). Benchmarking Cloud Serving Systems with YCSB. In: *SoCC*.

[40] **Das**, **A.**, **Hassan**, **H.**, and **Mutlu**, **O.** (2018). VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency. In: *DAC*.

[41] **Dennard**, **R. H.** (1968). Field-Effect Transistor Memory.

[42] **Devaux**, **F.** and **Ayrignac**, **R.** (2021). Method and Circuit for Protecting a DRAM Memory Device from the Row Hammer Effect. US Patent: 10,885,966.

[43] **Di Dio**, **A.**, **Koning**, **K.**, **Bos**, **H.**, and **Giuffrida**, **C.** (2023). Copy-on-Flip: Hardening ECC Memory Against Rowhammer Attacks. In: *NDSS*.

[44] **Enomoto**, **S.**, **Kuzuno**, **H.**, and **Yamada**, **H.** (2022). Efficient Protection Mechanism for CPU Cache Flush Instruction Based Attacks. In: *IEICE Transactions on Information and Systems*.

[45] **Fahr**, **M. J.** (2022). The Effects of Side-Channel Attacks on Post-Quantum Cryptography: Influencing FrodoKEM Key Generation Using the Rowhammer Exploit. PhD thesis. University of Arkansas.

[46] **Fahr Jr**, **M.**, **Kippen**, **H.**, **Kwong**, **A.**, **Dang**, **T.**, **Lichtinger**, **J.**, **Dachman-Soled**, **D.**, **Genkin**, **D.**, **Nelson**, **A.**, **Perlner**, **R.**, **Yerukhimovich**, **A.**, et al. (2022). When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer. In: *CCS*.

[47] **Fakhrzadehgan**, **A.**, **Patt**, **Y. N.**, **Nair**, **P. J.**, and **Qureshi**, **M. K.** (2022). SafeGuard: Reducing the Security Risk from Row-Hammer via Low-Cost Integrity Protection. In: *HPCA*.

[48] **Fournaris**, **A. P.**, **Pocero Fraile**, **L.**, and **Koufopavlou**, **O.** (2017). Exploiting Hardware Vulnerabilities to Attack Embedded System Devices: A Survey of Potent Microarchitectural Attacks. In: *Electronics*.

[49] **France**, **L.**, **Bruguier**, **F.**, **Mushtaq**, **M.**, **Novo**, **D.**, and **Benoit**, **P.** (2022). Modeling Rowhammer in the gem5 Simulator. In: *CHES 2022-Conference on Cryptographic Hardware and Embedded Systems*.

[50] **France**, **L.**, **Bruguier**, **F.**, **Novo**, **D.**, **Mushtaq**, **M.**, and **Benoit**, **P.** (2022). Reducing the Silicon Area Overhead of Counter-Based Rowhammer Mitigations. In: *18th CryptArchi Workshop*.

[51] **Frigo**, **P.**, **Giuffrida**, **C.**, **Bos**, **H.**, and **Razavi**, **K.** (2018). Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU. In: *S&P*.

[52] **Frigo**, **P.**, **Vannacci**, **E.**, **Hassan**, **H.**, **Veen**, **V. van der**, **Mutlu**, **O.**, **Giuffrida**, **C.**, **Bos**, **H.**, and **Razavi**, **K.** (2020). TRRespass: Exploiting the Many Sides of Target Row Refresh. In: *S&P*.

[53] **Fritts**, **J. E.**, **Steiling**, **F. W.**, **Tucek**, **J. A.**, and **Wolf**, **W.** (2009). MediaBench II Video: Expediting the next Generation of Video Systems Research. In: *Microprocess. Microsyst.*

[54] **Gao**, **F.**, **Tziantzioulis**, **G.**, and **Wentzlaff**, **D.** (2019). ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs. In: *MICRO*.

[55] — (2022). FracDRAM: Fractional Values in Off-the-Shelf DRAM. In: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[56] **Gautam**, **S.**, **Manhas**, **S.**, **Kumar**, **A.**, **Pakala**, **M.**, and **Yieh**, **E.** (2019). Row Hammering Mitigation Using Metal Nanowire in Saddle Fin DRAM. In: *IEEE TED*.

[57] **Genssler**, **P. R.**, **Santen**, **V. M. van**, **Henkel**, **J.**, and **Amrouch**, **H.** (2022). On the Reliability of FeFET On-Chip Memory. In: *TC*.

[58] **Gomez**, **H.**, **Amaya**, **A.**, and **Roa**, **E.** (2016). DRAM Row-Hammer Attack Reduction Using Dummy Cells. In: *NORCAS*.

[59] **Greenfield**, **Z.** and **Levy**, **T.** (2016). Throttling Support for Row-Hammer Counters.

[60] **Group**, **S. R.** (n.d.). Ramulator V2.0. `https://github.com/CMU-SAFARI/ramulator2`.

[61] **Gruss**, **D.**, **Lipp**, **M.**, **Schwarz**, **M.**, **Genkin**, **D.**, **Juffinger**, **J.**, **O'Connell**, **S.**, **Schoechl**, **W.**, and **Yarom**, **Y.** (2018). Another Flip in the Wall of Rowhammer Defenses. In: *S&P*.

[62] **Gruss**, **D.**, **Maurice**, **C.**, and **Mangard**, **S.** (2016). Rowhammer.js: A Remote Software-Induced Fault Attack in Javascript. arXiv:1507.06955 [cs.CR].

[63] **Gude Ramarao**, **C**, **Kumar**, **K. T.**, **Ujjinappa**, **G**, and **Naidu**, **B. V. D.** (2023). Defending SoCs with FPGAs from Rowhammer Attacks. In: *Material Science*.

[64] **Guha**, **K.** and **Chakrabarti**, **A.** (2022). Criticality based Reliability from Rowhammer Attacks in Multi-User-Multi-FPGA Platform. In: *VLSID*.

[65] **Hamming**, **R. W.** (1950). Error Detecting and Error Correcting Codes. In: *The Bell system technical journal*.

[66] **Han**, **J.**, **Kim**, **J.**, **Beery**, **D.**, **Bozdag**, **K. D.**, **Cuevas**, **P.**, **Levi**, **A.**, **Tain**, **I.**, **Tran**, **K.**, **Walker**, **A. J.**, **Palayam**, **S. V.**, et al. (2021). Surround Gate Transistor With Epitaxially Grown Si Pillar and Simulation Study on Soft Error and Rowhammer Tolerance for DRAM. In: *TED*.

[67] **Hassan**, **H.**, **Patel**, **M.**, **Kim**, **J. S.**, **Yağlıkçı**, **A. G.**, **Vijaykumar**, **N.**, **Mansouri Ghiasi**, **N.**, **Ghose**, **S.**, and **Mutlu**, **O.** (2019). CROW: A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability. In: *ISCA*.

[68] **Hassan**, **H.**, **Olgun**, **A.**, **Yaglikci**, **A. G.**, **Luo**, **H.**, and **Mutlu**, **O.** (2022). A Case for Self-Managing DRAM Chips: Improving Performance, Efficiency, Reliability, and Security via Autonomous in-DRAM Maintenance Operations. arXiv:2207.13358.

[69] **Hassan**, **H.**, **Pekhimenko**, **G.**, **Vijaykumar**, **N.**, **Seshadri**, **V.**, **Lee**, **D.**, **Ergin**, **O.**, and **Mutlu**, **O.** (2016). ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality. In: *HPCA*.

[70] **Hassan**, **H.**, **Tugrul**, **Y. C.**, **Kim**, **J. S.**, **Veen**, **V. v. d.**, **Razavi**, **K.**, and **Mutlu**, **O.** (2021). Uncovering in-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications. In: *MICRO*.

[71] **Hassan**, **H.**, **Vijaykumar**, **N.**, **Khan**, **S.**, **Ghose**, **S.**, **Chang**, **K.**, **Pekhimenko**, **G.**, **Lee**, **D.**, **Ergin**, **O.**, and **Mutlu**, **O.** (2017). SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies. In: *HPCA*.

[72] **He**, **W.**, **Zhang**, **Z.**, **Cheng**, **Y.**, **Wang**, **W.**, **Song**, **W.**, **Gao**, **Y.**, **Zhang**, **Q.**, **Li**, **K.**, **Liu**, **D.**, and **Nepal**, **S.** (2023). WhistleBlower: A System-level Empirical Study on RowHammer. In: *IEEE Transactions on Computers*.

[73] **Hewlett-Packard Enterprise** (2015). HP Moonshot Component Pack Version 2015.05.0.

[74] **Hocquenghem**, **A.** (1959). Codes Correcteurs d'Erreurs. In: *Chiffers*.

[75] **Hong**, **S.**, **Frigo**, **P.**, **Kaya**, **Y.**, **Giuffrida**, **C.**, and **Dumitraş**, **T.** (2019). Terminal Brain Damage: Exposing the Graceless Degradation in Deep

Neural Networks Under Hardware Fault Attacks. In: *USENIX Security*.

[76] **Hong**, **S.**, **Kim**, **D.**, **Lee**, **J.**, **Oh**, **R.**, **Yoo**, **C.**, **Hwang**, **S.**, and **Lee**, **J.** (2023). DSAC: Low-Cost Rowhammer Mitigation Using In-DRAM Stochastic and Approximate Counting Algorithm. arXiv:2302.03591.

[77] **Horiguchi**, **M.** (1997). Redundancy Techniques for High-Density DRAMs. In: *ISIS*.

[78] **Irazoqui**, **G.**, **Eisenbarth**, **T.**, and **Sunar**, **B.** (2016). MASCAT: Stopping Microarchitectural Attacks Before Execution. In: *IACR Cryptology*.

[79] **Islam**, **S.**, **Mus**, **K.**, **Singh**, **R.**, **Schaumont**, **P.**, and **Sunar**, **B.** (2022). Signature Correction Attack on Dilithium Signature Scheme. In: *Euro S&P*.

[80] **Itoh**, **K.** (2001). VLSI Memory Chip Design. Springer.

[81] **Jang**, **Y.**, **Lee**, **J.**, **Lee**, **S.**, and **Kim**, **T.** (2017). SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In: *SOSP*.

[82] **Jattke**, **P.**, **Veen**, **V. van der**, **Frigo**, **P.**, **Gunter**, **S.**, and **Razavi**, **K.** (2022). Blacksmith: Scalable Rowhammering in the Frequency Domain. In: *S&P*.

[83] **JEDEC** (2012a). *Standard No. 21C. Annex K: Serial Presence Detect (SPD) for DDR3 SDRAM Modules*.

[84] — (2012b). *Standard No. 79-3F. DDR3 SDRAM Specification*.

[85] — (2017). *JESD209-4B: Low Power Double Data Rate 4 (LPDDR4) Standard*.

[86] — (2020a). *JESD209-5A: LPDDR5 SDRAM Standard*.

[87] — (2020b). *JESD235C: High Bandwidth Memory (HBM) DRAM*.

[88] — (2020c). *JESD79-4C: DDR4 SDRAM Standard*.

[89] — (2020d). *JESD79-5: DDR5 SDRAM Standard*.

[90] — (2021). *JESD250C: Graphics Double Data Rate 6 (GDDR6) Standard*.

[91] — (2024). *JESD79-5c: DDR5 SDRAM Standard*.

[92] **Ji**, **S.**, **Ko**, **Y.**, **Oh**, **S.**, and **Kim**, **J.** (2019). Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks. In: *ASIACCS*.

[93] **Jiang**, **Y.**, **Zhu**, **H.**, **Sullivan**, **D.**, **Guo**, **X.**, **Zhang**, **X.**, and **Jin**, **Y.** (2021). Quantifying RowHammer Vulnerability for DRAM Security. In: *DAC*.

[94] **Joardar**, **B. K.**, **Bletsch**, **T. K.**, and **Chakrabarty**, **K.** (2022a). Learning to Mitigate RowHammer Attacks. In: *DATE*.

[95] **Joardar**, **B. K.**, **Bletsch**, **T. K.**, and **Chakrabarty**, **K.** (2022b). Machine Learning-based Rowhammer Mitigation. In: *TCAD*.

[96] **Juffinger**, **J.**, **Lamster**, **L.**, **Kogler**, **A.**, **Eichlseder**, **M.**, **Lipp**, **M.**, and **Gruss**, **D.** (2023). CSI: Rowhammer–Cryptographic Security and Integrity against Rowhammer (to appear). In: *S&P*.

[97] **Kang**, **I.**, **Lee**, **E.**, and **Ahn**, **J. H.** (2020). CAT-TWO: Counter-Based Adaptive Tree, Time Window Optimized for DRAM Row-Hammer Prevention. In: *IEEE Access*.

[98] **Kaseridis**, **D.**, **Stuecheli**, **J.**, and **John**, **L. K.** (2011). Minimalist Open-Page: A DRAM Page-Mode Scheduling Policy for the Many-Core Era. In: *MICRO*.

[99] **Kaur**, **A.**, **Srivastav**, **P.**, and **Ghoshal**, **B.** (2022). Work-in-Progress: DRAM-MaUT: DRAM Address Mapping Unveiling Tool for ARM Devices. In: *CASES*.

[100] **Keeth**, **B.** and **Baker**, **R.** (2001). DRAM Circuit Design: A Tutorial. John Wiley & Sons.

[101] **Khan**, **M. N. I.** and **Ghosh**, **S.** (2018). Analysis of Row Hammer Attack on STTRAM. In: *ICCD*.

[102] **Khan**, **S.**, **Lee**, **D.**, **Kim**, **Y.**, **Alameldeen**, **A. R.**, **Wilkerson**, **C.**, and **Mutlu**, **O.** (2014). The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study. In: *SIGMETRICS*.

[103] **Khan**, **S.**, **Lee**, **D.**, and **Mutlu**, **O.** (2016). PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM. In: *DSN*.

[104] **Khan**, **S.**, **Wilkerson**, **C.**, **Lee**, **D.**, **Alameldeen**, **A. R.**, and **Mutlu**, **O.** (2016). A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM. In: *CAL*.

[105] **Khan**, **S.**, **Wilkerson**, **C.**, **Wang**, **Z.**, **Alameldeen**, **A. R.**, **Lee**, **D.**, and **Mutlu**, **O.** (2017). Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content. In: *MICRO*.

[106] **Kim**, **D.-H.**, **Nair**, **P. J.**, and **Qureshi**, **M. K.** (2014). Architectural Support for Mitigating Row Hammering in DRAM Memories. In: *CAL*.

[107] **Kim**, **J. S.**, **Patel**, **M.**, **Hassan**, **H.**, and **Mutlu**, **O.** (2018a). Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines. In: *ICCD*.

[108] **Kim**, **J. S.**, **Patel**, **M.**, **Hassan**, **H.**, and **Mutlu**, **O.** (2018b). The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern Commodity DRAM Devices. In: *HPCA*.

[109] **Kim**, **J. S.**, **Patel**, **M.**, **Hassan**, **H.**, **Orosa**, **L.**, and **Mutlu**, **O.** (2019). D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput. In: *HPCA*.

[110] **Kim**, **J. S.**, **Patel**, **M.**, **Yağlıkçı**, **A. G.**, **Hassan**, **H.**, **Azizi**, **R.**, **Orosa**, **L.**, and **Mutlu**, **O.** (2020). Revisiting RowHammer: An Experimental Analysis of Modern Devices and Mitigation Techniques. In: *ISCA*.

[111] **Kim**, **J.**, **Sullivan**, **M.**, and **Erez**, **M.** (2015). Bamboo ECC: Strong, safe, and flexible codes for reliable computer memory. In: *HPCA*.

[112] **Kim**, **M. J.**, **Park**, **J.**, **Park**, **Y.**, **Doh**, **W.**, **Kim**, **N.**, **Ham**, **T. J.**, **Lee**, **J. W.**, and **Ahn**, **J. H.** (2022). Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh. In: *HPCA*.

[113] **Kim**, **W.**, **Jung**, **C.**, **Yoo**, **S.**, **Hong**, **D.**, **Hwang**, **J.**, **Yoon**, **J.**, **Jung**, **O.**, **Choi**, **J.**, **Hyun**, **S.**, **Kang**, **M.**, et al. (2023). A 1.1 V 16Gb DDR5 DRAM with Probabilistic-Aggressor Tracking, Refresh-Management Functionality, Per-Row Hammer Tracking, a Multi-Step Precharge, and Core-Bias Modulation for Security and Reliability Enhancement. In: *ISSCC*.

[114] **Kim**, **Y.**, **Daly**, **R.**, **Kim**, **J.**, **Fallin**, **C.**, **Lee**, **J. H.**, **Lee**, **D.**, **Wilkerson**, **C.**, **Lai**, **K.**, and **Mutlu**, **O.** (2014). Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In: *ISCA*.

[115] **Kim**, **Y.**, **Seshadri**, **V.**, **Lee**, **D.**, **Liu**, **J.**, **Mutlu**, **O.**, **Kim**, **Y.**, **Seshadri**, **V.**, **Lee**, **D.**, **Liu**, **J.**, and **Mutlu**, **O.** (2012). A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM. In: *ISCA*.

[116] **Kogler**, **A.**, **Juffinger**, **J.**, **Qazi**, **S.**, **Kim**, **Y.**, **Lipp**, **M.**, **Boichat**, **N.**, **Shiu**, **E.**, **Nissler**, **M.**, and **Gruss**, **D.** (2022). Half-Double: Hammering From the Next Row Over. In: *USENIX Security*.

[117] **Konoth**, **R. K.**, **Oliverio**, **M.**, **Tatar**, **A.**, **Andriesse**, **D.**, **Bos**, **H.**, **Giuffrida**, **C.**, and **Razavi**, **K.** (2018). ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks. In: *OSDI*.

[118] **Koppula**, **S.**, **Orosa**, **L.**, **Yağlıkçı**, **A. G.**, **Azizi**, **R.**, **Shahroodi**, **T.**, **Kanellopoulos**, **K.**, and **Mutlu**, **O.** (2019). EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM. In: *MICRO*.

[119] **Kwong**, **A.**, **Genkin**, **D.**, **Gruss**, **D.**, and **Yarom**, **Y.** (2020). RAMBleed: Reading Bits in Memory Without Accessing Them. In: *S&P*.

[120] **Lee**, **D.**, **Khan**, **S.**, **Subramanian**, **L.**, **Ghose**, **S.**, **Ausavarungnirun**, **R.**, **Pekhimenko**, **G.**, **Seshadri**, **V.**, and **Mutlu**, **O.** (2017). Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms. In: *SIGMETRICS*.

[121] **Lee**, **D.**, **Kim**, **Y.**, **Pekhimenko**, **G.**, **Khan**, **S.**, **Seshadri**, **V.**, **Chang**, **K.**, and **Mutlu**, **O.** (2015). Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case. In: *HPCA*.

[122] **Lee**, **D.**, **Kim**, **Y.**, **Seshadri**, **V.**, **Liu**, **J.**, **Subramanian**, **L.**, and **Mutlu**, **O.** (2013). Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture. In: *HPCA*.

[123] **Lee**, **E.**, **Kang**, **I.**, **Lee**, **S.**, **Edward Suh**, **G**, and **Ho Ahn**, **J.** (2019). TWiCe: Preventing Row-Hammering by Exploiting Time Window Counters. In: *ISCA*.

[124] **Lee**, **G.-H.**, **Na**, **S.**, **Byun**, **I.**, **Min**, **D.**, and **Kim**, **J.** (2021). CryoGuard: A Near Refresh-Free Robust DRAM Design for Cryogenic Computing. In: *ISCA*.

[125] **Lefforge**, **S.** (2023). Reverse Engineering Post-Quantum Cryptography Schemes to Find Rowhammer Exploits. MA thesis. University of Arkansas.

[126] **Lenovo Group Ltd.** (2015). Row Hammer Privilege Escalation.

[127] **Li**, **D.**, **Liu**, **D.**, **Ren**, **Y.**, **Wang**, **Z.**, **Sun**, **Y.**, **Guan**, **Z.**, **Wu**, **Q.**, and **Liu**, **J.** (2022). CyberRadar: A PUF-based Detecting and Mapping Framework for Physical Devices. arXiv:2201.07597.

[128] **Li**, **H.**, **Chen**, **H.-Y.**, **Chen**, **Z.**, **Chen**, **B.**, **Liu**, **R.**, **Qiu**, **G.**, **Huang**, **P.**, **Zhang**, **F.**, **Jiang**, **Z.**, **Gao**, **B.**, **Liu**, **L.**, **Liu**, **X.**, **Yu**, **S.**, **Wong**, **H.-S. P.**, and **Kang**, **J.** (2014). Write Disturb Analyses on Half-Selected Cells of Cross-Point RRAM Arrays. In: *IRPS*.

[129] **Lim**, **C.**, **Park**, **K.**, and **Baeg**, **S.** (2017). Active Precharge Hammering to Monitor Displacement Damage Using High-Energy Protons in 3x-nm SDRAM. In: *TNS*.

[130] **Lim**, **C.**, **Park**, **K.**, **Bak**, **G.**, **Yun**, **D.**, **Park**, **M.**, **Baeg**, **S.**, **Wen**, **S.-J.**, and **Wong**, **R.** (2018). Study of Proton Radiation Effect to Row Hammer Fault in DDR4 SDRAMs. In: *Microelectronics Reliability*.

[131] **Lipp**, **M.**, **Aga**, **M. T.**, **Schwarz**, **M.**, **Gruss**, **D.**, **Maurice**, **C.**, **Raab**, **L.**, and **Lamster**, **L.** (2018). Nethammer: Inducing Rowhammer Faults Through Network Requests. arXiv:1805.04956 [cs.CR].

[132] **Liu**, **J.**, **Jaiyen**, **B.**, **Kim**, **Y.**, **Wilkerson**, **C.**, **Mutlu**, **O.**, **Liu**, **J**, **Jaiyen**, **B**, **Kim**, **Y**, **Wilkerson**, **C**, and **Mutlu**, **O** (2013). An Experimental Study of Data Retention Behavior in Modern DRAM Devices. In: *ISCA*.

[133] **Liu**, **L.**, **Guo**, **Y.**, **Cheng**, **Y.**, **Zhang**, **Y.**, and **Yang**, **J.** (2022). Generating Robust DNN with Resistance to Bit-Flip based Adversarial Weight Attack. In: *IEEE TC*.

[134] **Loughlin**, **K.**, **Saroiu**, **S.**, **Wolman**, **A.**, and **Kasikci**, **B.** (2021). Stop! Hammer Time: Rethinking Our Approach to Rowhammer Mitigations. In: *HotOS*.

[135] **Loughlin**, **K.**, **Saroiu**, **S.**, **Wolman**, **A.**, **Manerkar**, **Y. A.**, and **Kasikci**, **B.** (2022). MOESI-Prime: Preventing Coherence-Induced Hammering in Commodity Workloads. In: *ISCA*.

[136] **Luo**, **H.**, **Olgun**, **A.**, **Yağlıkçı**, **A. G.**, **Tuğrul**, **Y. C.**, **Rhyner**, **S.**, **Cavlak**, **M. B.**, **Lindegger**, **J.**, **Sadrosadati**, **M.**, and **Mutlu**, **O.** (2023). RowPress: Amplifying Read Disturbance in Modern DRAM Chips. In: *ISCA*.

[137] **Luo**, **H.**, **Shahroodi**, **T.**, **Hassan**, **H.**, **Patel**, **M.**, **Yaglikci**, **A. G.**, **Orosa**, **L.**, **Park**, **J.**, and **Mutlu**, **O.** (2020). CLR-DRAM: A Low-Cost DRAM Architecture Enabling Dynamic Capacity-Latency Trade-Off. In: *ISCA*.

[138] **Luo**, **H.**, **Tuğrul**, **Y. C.**, **Bostancı**, **F. N.**, **Olgun**, **A.**, **Yağlıkçı**, **A. G.**, and **Mutlu**, **O.** (2023). Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator. arXiv:2308.11030v1 [cs.AR].

[139] **Manzhosov**, **E.**, **Hastings**, **A.**, **Pancholi**, **M.**, **Piersma**, **R.**, **Ziad**, **M. T. I.**, and **Sethumadhavan**, **S.** (2022). Revisiting Residue Codes for Modern Memories. In: *MICRO*.

[140] **Marazzi**, **M.**, **Jattke**, **P.**, **Solt**, **F.**, and **Razavi**, **K.** (2022). ProTRR: Principled yet Optimal In-DRAM Target Row Refresh. In: *S&P*.

[141] **Marazzi**, **M.**, **Solt**, **F.**, **Jattke**, **P.**, **Takashi**, **K.**, and **Razavi**, **K.** (2023). REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations. In: *S&P*.

[142] **Mathew**, **D. M.**, **Zulian**, **E. F.**, **Jung**, **M.**, **Kraft**, **K.**, **Weis**, **C.**, **Jacob**, **B.**, and **Wehn**, **N.** (2017). Using Run-Time Reverse-Engineering to Optimize DRAM Refresh. In: *MEMSYS*.

[143] **Maxwell** (n.d.). FT20X User Manual. `https://www.maxwell-fa.com/upload/files/base/8/m/311.pdf`.

[144] **Mukhanov**, **L.**, **Nikolopoulos**, **D. S.**, and **Karakonstantis**, **G.** (2020). DStress: Automatic Synthesis of DRAM Reliability Stress Viruses using Genetic Algorithms (Best Paper Nominee). In: *MICRO*.

[145] **Mus**, **K.**, **Doröz**, **Y.**, **Tol**, **M. C.**, **Rahman**, **K.**, and **Sunar**, **B.** (2022). Jolt: Recovering TLS Signing Keys via Rowhammer Faults. In: *Cryptology ePrint Archive*.

[146] **Mutlu**, **O.** (2017). The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser. In: *DATE*.

[147] — (2018). RowHammer. `https://people.inf.ethz.ch/omutlu/pub/onur-Rowhammer-TopPicksinHardwareEmbeddedSecurity-November-8-2018.pdf`.

[148] **Mutlu**, **O.** and **Kim**, **J. S.** (2019). RowHammer: A Retrospective. In: *TCAD*.

[149] **Mutlu**, **O.**, **Olgun**, **A.**, and **Yaglikci**, **A. G.** (2023). Fundamentally Understanding and Solving RowHammer. In: *ASP-DAC*.

[150] **Nair**, **P. J.**, **Sridharan**, **V.**, and **Qureshi**, **M. K.** (2016). XED: Exposing on-die error detection information for strong memory reliability. In: *ISCA*.

[151] **Nale**, **B.** and **Bains**, **K. S.** (2021). Perfect row hammer tracking with multiple count increments. US Patent: US20220121398A1.

[152] **Naseredini**, **A.**, **Berger**, **M.**, **Sammartino**, **M.**, and **Xiong**, **S.** (2022). ALARM: Active LeArning of Rowhammer Mitigations. `https://users.sussex.ac.uk/~mfb21/rh-draft.pdf`.

[153] **Ni**, **K.**, **Li**, **X.**, **Smith**, **J. A.**, **Jerry**, **M.**, and **Datta**, **S.** (2018). Write Disturb in Ferroelectric FETs and Its Implication for 1T-FeFET AND Memory Arrays. In: *IEEE EDL*.

[154] **Olgun**, **A.**, **Hassan**, **H.**, **Yağlıkcı**, **A. G.**, **Tuğrul**, **Y. C.**, **Orosa**, **L.**, **Luo**, **H.**, **Patel**, **M.**, **Oğuz**, **E.**, and **Mutlu**, **O.** (2023). DRAM Bender: An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips. In: *TCAD*.

[155] **Olgun**, **A.**, **Luna**, **J. G.**, **Kanellopoulos**, **K.**, **Salami**, **B.**, **Hassan**, **H.**, **Ergin**, **O.**, and **Mutlu**, **O.** (2022). PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM. In: *ACM Transactions on Architecture and Code Optimization*.

[156] **Olgun**, **A.**, **Osseiran**, **M.**, **Yaglikci**, **A. G.**, **Tugrul**, **Y. C.**, **Luo**, **H.**, **Rhyner**, **S.**, **Salami**, **B.**, **Gomez Luna**, **J.**, and **Mutlu**, **O.** (2023). An

Experimental Analysis of RowHammer in HBM2 DRAM Chips. In: *DSN Disrupt*.

[157] **Olgun**, **A.**, **Patel**, **M.**, **Yağlıkçı**, **A. G.**, **Luo**, **H.**, **Kim**, **J. S.**, **Bostancı**, **N.**, **Vijaykumar**, **N.**, **Ergin**, **O.**, and **Mutlu**, **O.** (2021). QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips. In: *ISCA*.

[158] **Orosa**, **L.**, **Koppula**, **S.**, **Kanellopoulos**, **K.**, **Yağlıkçı**, **A. G.**, and **Mutlu**, **O.** (Oct. 2023). "Using Approximate DRAM for Enabling Energy-Efficient, High-Performance Deep Neural Network Inference". In: Springer, pp. 275–314.

[159] **Orosa**, **L.**, **Rührmair**, **U.**, **Yaglikci**, **A. G.**, **Luo**, **H.**, **Olgun**, **A.**, **Jattke**, **P.**, **Patel**, **M.**, **Kim**, **J.**, **Razavi**, **K.**, and **Mutlu**, **O.** (2022). SpyHammer: Using RowHammer to Remotely Spy on Temperature. arXiv:2210.04084.

[160] **Orosa**, **L.**, **Wang**, **Y.**, **Sadrosadati**, **M.**, **Kim**, **J. S.**, **Patel**, **M.**, **Puddu**, **I.**, **Luo**, **H.**, **Razavi**, **K.**, **Gómez-Luna**, **J.**, **Hassan**, **H.**, **Mansouri-Ghiasi**, **N.**, **Ghose**, **S.**, and **Mutlu**, **O.** (2021). CODIC: A Low-Cost Substrate for Enabling Custom In-DRAM Functionalities and Optimizations. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*.

[161] **Orosa**, **L.**, **Yağlıkçı**, **A. G.**, **Luo**, **H.**, **Olgun**, **A.**, **Park**, **J.**, **Hassan**, **H.**, **Patel**, **M.**, **Kim**, **J. S.**, and **Mutlu**, **O.** (2021). A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses. In: *MICRO*.

[162] **Park**, **J. H.**, **Kim**, **S. Y.**, **Kim**, **D. Y.**, **Kim**, **G.**, **Park**, **J. W.**, **Yoo**, **S.**, **Lee**, **Y.-W.**, and **Lee**, **M. J.** (2022). RowHammer Reduction Using a Buried Insulator in a Buried Channel Array Transistor. In: *IEEE Transactions on Electron Devices*.

[163] **Park**, **K.**, **Baeg**, **S.**, **Wen**, **S.**, and **Wong**, **R.** (2014). Active-Precharge Hammering on a Row-Induced Failure in DDR3 SDRAMs Under 3x nm Technology. In: *IIRW*.

[164] **Park**, **K.**, **Lim**, **C.**, **Yun**, **D.**, and **Baeg**, **S.** (2016). Experiments and Root Cause Analysis for Active-Precharge Hammering Fault in DDR3 SDRAM under 3xnm Technology. In: *Microelectronics Reliability*.

[165] **Park**, **K.**, **Yun**, **D.**, and **Baeg**, **S.** (2016). Statistical Distributions of Row-Hammering Induced Failures in DDR3 Components. In: *Microelectronics Reliability*.

[166] **Park**, **Y.**, **Kwon**, **W.**, **Lee**, **E.**, **Ham**, **T. J.**, **Ahn**, **J. H.**, and **Lee**, **J. W.** (2020). Graphene: Strong yet Lightweight Row Hammer Protection. In: *MICRO*.

[167] **Patel**, **M.**, **Kim**, **J.**, **Shahroodi**, **T.**, **Hassan**, **H.**, and **Mutlu**, **O.** (2020). Bit-Exact ECC Recovery (BEER): Determining DRAM On-Die ECC

Functions by Exploiting DRAM Data Retention Characteristics (Best Paper). In: *MICRO*.

[168] **Patel**, **M.**, **Shahroodi**, **T.**, **Manglik**, **A.**, **Yaglikci**, **A. G.**, **Olgun**, **A.**, **Luo**, **H.**, and **Mutlu**, **O.** (2022). A Case for Transparent Reliability in DRAM Systems. arXiv:2204.10378.

[169] **Pessl**, **P.**, **Gruss**, **D.**, **Maurice**, **C.**, **Schwarz**, **M.**, and **Mangard**, **S.** (2016). DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In: *USENIX Security*.

[170] **Poddebniak**, **D.**, **Somorovsky**, **J.**, **Schinzel**, **S.**, **Lochter**, **M.**, and **Rösler**, **P.** (2018). Attacking Deterministic Signature Schemes using Fault Attacks. In: *EuroS&P*.

[171] **Qiao**, **R.** and **Seaborn**, **M.** (2016). A New Approach for RowHammer Attacks. In: *HOST*.

[172] **Qin**, **Y.**, **Lin**, **C.**, **He**, **W.**, **Sun**, **Y.**, **Mao**, **Z.**, and **Seok**, **M.** (2023). CDAR-DRAM: Enabling Runtime DRAM Performance and Energy Optimization via In-situ Charge Detection and Adaptive Data Restoration. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[173] **Qureshi**, **M.** (2021). Rethinking ECC in the Era of Row-Hammer. In: *DRAMSec*.

[174] **Qureshi**, **M.**, **Rohan**, **A.**, **Saileshwar**, **G.**, and **Nair**, **P. J.** (2022). Hydra: Enabling Low-Overhead Mitigation of Row-Hammer at Ultra-Low Thresholds via Hybrid Tracking. In: *ISCA*.

[175] **Rakin**, **A. S.**, **Chowdhuryy**, **M. H. I.**, **Yao**, **F.**, and **Fan**, **D.** (2022). DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories. In: *S&P*.

[176] **Razavi**, **K.**, **Gras**, **B.**, **Bosman**, **E.**, **Preneel**, **B.**, **Giuffrida**, **C.**, and **Bos**, **H.** (2016). Flip Feng Shui: Hammering a Needle in the Software Stack. In: *USENIX Security*.

[177] **Redeker**, **M.**, **Cockburn**, **B. F.**, and **Elliott**, **D. G.** (2002). An Investigation into Crosstalk Noise in DRAM Structures. In: *MTDT*.

[178] **Ridder**, **F. de**, **Frigo**, **P.**, **Vannacci**, **E.**, **Bos**, **H.**, **Giuffrida**, **C.**, and **Razavi**, **K.** (2021). SMASH: Synchronized Many-Sided Rowhammer Attacks from JavaScript. In: *USENIX Security*.

[179] **Rixner**, **S.**, **Dally**, **W. J.**, **Kapasi**, **U. J.**, **Mattson**, **P.**, and **Owens**, **J. D.** (2000). Memory Access Scheduling. In: *ISCA*.

[180] **Roohi**, **A.** and **Angizi**, **S.** (2022). Efficient Targeted Bit-Flip Attack Against the Local Binary Pattern Network. In: *HOST*.

[181] **Ryu**, **S.-W.**, **Min**, **K.**, **Shin**, **J.**, **Kwon**, **H.**, **Nam**, **D.**, **Oh**, **T.**, **Jang**, **T.-S.**, **Yoo**, **M.**, **Kim**, **Y.**, and **Hong**, **S.** (2017). Overcoming the Reliability Limitation in the Ultimately Scaled DRAM using Silicon Migration Technique by Hydrogen Annealing. In: *IEDM*.

[182] **SAFARI Research Group** (2014). RowHammer — GitHub Repository. `https://github.com/CMU-SAFARI/rowhammer`.

[183] — (2017). SoftMC — GitHub Repository. `https://github.com/CMU-SAFARI/softmc`.

[184] — (2022). DRAM Bender — GitHub Repository. https://github.com/CMU-SAFARI/DRAM-Bender.

[185] **Saileshwar**, **G.**, **Wang**, **B.**, **Qureshi**, **M.**, and **Nair**, **P. J.** (2022). Randomized Row-Swap: Mitigating Row Hammer by Breaking Spatial Correlation Between Aggressor and Victim Rows. In: *ASPLOS*.

[186] **Samsung** (2017). 288pin Registered DIMM based on 8Gb B-die, Rev 1.91. https://semiconductor.samsung.com/resources/data-sheet/20170731_DDR4_8Gb_B_die_Registered_DIMM_Rev1.91_May.17.pdf.

[187] **Saroiu**, **S.** and **Wolman**, **A.** (2022). How to Configure Row-Sampling-Based Rowhammer Defenses. In: *DRAMSec*.

[188] **Saroiu**, **S.**, **Wolman**, **A.**, and **Cojocar**, **L.** (2022). The Price of Secrecy: How Hiding Internal DRAM Topologies Hurts Rowhammer Defenses. In: *IRPS*.

[189] **Saxena**, **A.**, **Saileshwar**, **G.**, **Juffinger**, **J.**, **Kogler**, **A.**, **Gruss**, **D.**, and **Qureshi**, **M.** (2023). PT-Guard: Integrity-Protected Page Tables to Defend Against Breakthrough Rowhammer Attacks. In: *DSN*.

[190] **Saxena**, **A.**, **Saileshwar**, **G.**, **Nair**, **P. J.**, and **Qureshi**, **M.** (2022). AQUA: Scalable Rowhammer Mitigation by Quarantining Aggressor Rows at Runtime. In: *MICRO*.

[191] **Seaborn**, **M.** and **Dullien**, **T.** (2015a). Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges. http://googleprojectzero.blogspot.com.tr/2015/03/exploiting-dram-rowhammer-bug-to-gain.html.

[192] — (2015b). Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges. In: *Black Hat*.

[193] **Seyedzadeh**, **S. M.**, **Jones**, **A. K.**, and **Melhem**, **R.** (2018). Mitigating Wordline Crosstalk Using Adaptive Trees of Counters. In: *ISCA*.

[194] **Seyedzadeh**, **S. M.**, **Jones**, **A. K.**, and **Melhem**, **R.** (2017). Counter-Based Tree Structure for Row Hammering Mitigation in DRAM. In: *CAL*.

[195] **Sharma**, **S.**, **Sanyal**, **D.**, **Mukhopadhyay**, **A.**, and **Shaik**, **R. H.** (2022). A Review on Study of Defects of DRAM-RowHammer and Its Mitigation. In: *Journal For Basic Sciences*.

[196] **Sherwood**, **T.**, **Perelman**, **E.**, **Hamerly**, **G.**, and **Calder**, **B.** (2002). Automatically Characterizing Large Scale Program Behavior. In: *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*.

[197] **Shin**, **W.**, **Yang**, **J.**, **Choi**, **J.**, and **Kim**, **L.-S.** (2014). NUAT: A Non-Uniform Access Time Memory Controller. In: *HPCA*.

[198] **Son**, **M.**, **Park**, **H.**, **Ahn**, **J.**, and **Yoo**, **S.** (2017). Making DRAM Stronger Against Row Hammering. In: *DAC*.

[199] **Standard Performance Evaluation Corp.** (n.d.[a]). SPEC CPU 2006. http://www.spec.org/cpu2006/.

[200] — (n.d.[b]). SPEC CPU2017 Benchmarks. http://www.spec.org/cpu2017/.

[201] **Staudigl**, **F.**, **Al Indari**, **H.**, **Schön**, **D.**, **Sisejkovic**, **D.**, **Merchant**, **F.**, **Joseph**, **J. M.**, **Rana**, **V.**, **Menzel**, **S.**, and **Leupers**, **R.** (2022). NeuroHammer: Inducing Bit-Flips in Memristive Crossbar Memories. In: *DATE*.

[202] **Subramanian**, **L.**, **Lee**, **D.**, **Seshadri**, **V.**, **Rastogi**, **H.**, and **Mutlu**, **O.** (2014). The Blacklisting Memory Scheduler: Achieving High Performance and Fairness at Low Cost. In: *ICCD*.

[203] **Tam**, **S. M.**, **Muljono**, **H.**, **Huang**, **M.**, **Iyer**, **S.**, **Royneogi**, **K.**, **Satti**, **N.**, **Qureshi**, **R.**, **Chen**, **W.**, **Wang**, **T.**, **Hsieh**, **H.**, **Vora**, **S.**, and **Wang**, **E.** (2018). SkyLake-SP: A 14nm 28-Core xeon® processor. In: *2018 IEEE International Solid-State Circuits Conference - (ISSCC)*, pp. 34–36.

[204] **Tatar**, **A.**, **Giuffrida**, **C.**, **Bos**, **H.**, and **Razavi**, **K.** (2018). Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer. In: *RAID*.

[205] **Tatar**, **A.**, **Konoth**, **R. K.**, **Athanasopoulos**, **E.**, **Giuffrida**, **C.**, **Bos**, **H.**, and **Razavi**, **K.** (2018). Throwhammer: Rowhammer Attacks Over the Network and Defenses. In: *USENIX ATC*.

[206] **Tobah**, **Y.**, **Kwong**, **A.**, **Kang**, **I.**, **Genkin**, **D.**, and **Shin**, **K. G.** (2022). SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks. In: *S&P*.

[207] **Tol**, **M. C.**, **Islam**, **S.**, **Sunar**, **B.**, and **Zhang**, **Z.** (2022). Toward Realistic Backdoor Injection Attacks on DNNs using RowHammer. arXiv:2110.07683.

[208] **Tomita**, **C.**, **Takita**, **M.**, **Fukushima**, **K.**, **Nakano**, **Y.**, **Shiraishi**, **Y.**, and **Morii**, **M.** (2022). Extracting the Secrets of OpenSSL with RAMBleed. In: *Sensors*.

[209] **Transaction Processing Performance Council** (n.d.). TPC Benchmarks. `http://tpc.org/`.

[210] **Veen**, **V. van der**, **Fratantonio**, **Y.**, **Lindorfer**, **M.**, **Gruss**, **D.**, **Maurice**, **C.**, **Vigna**, **G.**, **Bos**, **H.**, **Razavi**, **K.**, and **Giuffrida**, **C.** (2016). Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In: *CCS*.

[211] **Veen**, **V. van der**, **Lindorfer**, **M.**, **Fratantonio**, **Y.**, **Pillai**, **H. P.**, **Vigna**, **G.**, **Kruegel**, **C.**, **Bos**, **H.**, and **Razavi**, **K.** (2018). GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM. In: *DIMVA*.

[212] **Vig**, **S.**, **Bhattacharya**, **S.**, **Mukhopadhyay**, **D.**, and **Lam**, **S.-K.** (2018). Rapid Detection of Rowhammer Attacks Using Dynamic Skewed Hash Tree. In: *HASP*.

[213] **Walker**, **A. J.**, **Lee**, **S.**, and **Beery**, **D.** (2021). On DRAM RowHammer and the Physics on Insecurity. In: *IEEE TED*.

[214] **Wang**, **J.**, **Xu**, **H.**, **Xiao**, **C.**, **Zhang**, **L.**, and **Zheng**, **Y.** (2022). Research and Implementation of Rowhammer Attack Method based on Domestic NeoKylin Operating System. In: *ICFTIC*.

[215] **Wang**, **Y.**, **Tavakkol**, **A.**, **Orosa**, **L.**, **Ghose**, **S.**, **Ghiasi**, **N. M.**, **Patel**, **M.**, **Kim**, **J. S.**, **Hassan**, **H.**, **Sadrosadati**, **M.**, and **Mutlu**, **O.** (2018). Reducing DRAM Latency via Charge-Level-Aware Look-Ahead Partial Restoration. In: *MICRO*.

[216] **Weissman**, **Z.**, **Tiemann**, **T.**, **Moghimi**, **D.**, **Custodio**, **E.**, **Eisenbarth**, **T.**, and **Sunar**, **B.** (2020). JackHammer: Efficient Rowhammer on Heterogeneous FPGA–CPU Platforms. arXiv:1912.11523 [cs.CR].

[217] **Wi**, **M.**, **Park**, **J.**, **Ko**, **S.**, **Kim**, **M. J.**, **Kim**, **N. S.**, **Lee**, **E.**, and **Ahn**, **J. H.** (2023). SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling. In: *HPCA*.

[218] **WikiChip** (n.d.). Cascade Lake SP - Intel. `https://en.wikichip.org/wiki/intel/cores/cascade_lake_sp`.

[219] **Woo**, **J.**, **Saileshwar**, **G.**, and **Nair**, **P. J.** (2022). Scalable and Secure Row-Swap: Efficient and Safe Row Hammer Mitigation in Memory Systems. In.

[220] **Woo**, **J.**, **Saileshwar**, **G.**, and **Nair**, **P. J.** (2023). Scalable and Secure Row-Swap: Efficient and Safe Row Hammer Mitigation in Memory Systems. In: *HPCA*.

[221] **Xiao**, **Y.**, **Zhang**, **X.**, **Zhang**, **Y.**, and **Teodorescu**, **R.** (2016). One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In: *USENIX Security*.

[222] **Xilinx Inc.** (n.d.). Xilinx Alveo U200 FPGA Board. `https://www.xilinx.com/products/boards-and-kits/alveo/u200.html`.

[223] **Yağlıkçı**, **A. G.**, **Kim**, **J. S.**, **Devaux**, **F.**, and **Mutlu**, **O.** (2021). Security Analysis of the Silver Bullet Technique for RowHammer Prevention. arXiv:2106.07084.

[224] **Yağlıkcı**, **A. G.**, **Luo**, **H.**, **De Oliviera**, **G. F.**, **Olgun**, **A.**, **Patel**, **M.**, **Park**, **J.**, **Hassan**, **H.**, **Kim**, **J. S.**, **Orosa**, **L.**, and **Mutlu**, **O.** (2022). Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices. In: *DSN*.

[225] **Yağlikci**, **A. G.**, **Olgun**, **A.**, **Patel**, **M.**, **Luo**, **H.**, **Hassan**, **H.**, **Orosa**, **L.**, **Ergin**, **O.**, and **Mutlu**, **O.** (2022). HiRA: Hidden Row Activation for Reducing Refresh Latency of Off-the-Shelf DRAM Chips. In: *MICRO*.

[226] **Yağlıkçı**, **A. G.**, **Patel**, **M.**, **Kim**, **J. S.**, **Azizibarzoki**, **R.**, **Olgun**, **A.**, **Orosa**, **L.**, **Hassan**, **H.**, **Park**, **J.**, **Kanellopoullos**, **K.**, **Shahroodi**, **T.**, **Ghose**, **S.**, and **Mutlu**, **O.** (2021). BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows. In: *HPCA*.

[227] **Yang**, **C.**, **Wei**, **C. K.**, **Chang**, **Y. J.**, **Wu**, **T. C.**, **Chen**, **H. P.**, and **Lai**, **C. S.** (2016). Suppression of RowHammer Effect by Doping Profile Modification in Saddle-Fin Array Devices for Sub-30-nm DRAM Technology. In: *TDMR*.

[228] **Yang**, **C.-M.**, **Wei**, **C.-K.**, **Chen**, **H.-P.**, **Luo**, **J.-S.**, **Chang**, **Y. J.**, **Wu**, **T.-C.**, and **Lai**, **C.-S.** (2017). Scanning Spreading Resistance Microscopy

for Doping Profile in Saddle-Fin Devices. In: *IEEE Transactions on Nanotechnology*.

[229] **Yang**, **L.-H.**, **Huang**, **S.-S.**, **Cheng**, **T.-L.**, **Kuo**, **Y.-C.**, and **Kuo**, **J.-J.** (2022). Socially-Aware Collaborative Defense System against Bit-Flip Attack in Social Internet of Things and Its Online Assignment Optimization. In: *ICCCN*.

[230] **Yang**, **T.** and **Lin**, **X.-W.** (2019). Trap-Assisted DRAM Row Hammer Effect. In: *EDL*.

[231] **Yao**, **F.**, **Rakin**, **A. S.**, and **Fan**, **D.** (2020). Deephammer: Depleting the Intelligence of Deep Neural Networks Through Targeted Chain of Bit Flips. In: *USENIX Security*.

[232] **You**, **J. M.** and **Yang**, **J.-S.** (2019). MRLoc: Mitigating Row-Hammering Based on Memory Locality. In: *DAC*.

[233] **Yun**, **D.**, **Park**, **M.**, **Lim**, **C.**, and **Baeg**, **S.** (2018). Study of TID Effects on One Row Hammering using Gamma in DDR4 SDRAMs. In: *IRPS*.

[234] **Zhang**, **D.**, **Panwar**, **G.**, **Kotra**, **J. B.**, **DeBardeleben**, **N.**, **Blanchard**, **S.**, and **Jian**, **X.** (2021). Quantifying server memory frequency margin and using it to improve performance in HPC systems. In: *ISCA*. Virtual Event, Spain.

[235] **Zhang**, **X.**, **Zhang**, **Y.**, **Childers**, **B. R.**, and **Yang**, **J.** (2016). Restore Truncation for Performance Improvement in Future DRAM Systems. In: *HPCA*.

[236] **Zhang**, **Z.**, **Zhan**, **Z.**, **Balasubramanian**, **D.**, **Koutsoukos**, **X.**, and **Karsai**, **G.** (2018). Triggering Rowhammer Hardware Faults on ARM: A Revisit. In: *ASHES*.

[237] **Zhang**, **Z.**, **Zhan**, **Z.**, **Balasubramanian**, **D.**, **Li**, **B.**, **Volgyesi**, **P.**, and **Koutsoukos**, **X.** (2020). Leveraging EM Side-Channel Information to Detect Rowhammer Attacks. In: *S&P*.

[238] **Zhang**, **Z.**, **Cheng**, **Y.**, **Liu**, **D.**, **Nepal**, **S.**, **Wang**, **Z.**, and **Yarom**, **Y.** (2020). PThammer: Cross-User-Kernel-Boundary Rowhammer through Implicit Accesses. In: *MICRO*.

[239] **Zhang**, **Z.**, **Cheng**, **Y.**, **Wang**, **M.**, **He**, **W.**, **Wang**, **W.**, **Nepal**, **S.**, **Gao**, **Y.**, **Li**, **K.**, **Wang**, **Z.**, and **Wu**, **C.** (2022). SoftTRR: Protect Page Tables against Rowhammer Attacks using Software-only Target Row Refresh. In: *USENIX ATC*.

[240] **Zhang**, **Z.**, **He**, **W.**, **Cheng**, **Y.**, **Wang**, **W.**, **Gao**, **Y.**, **Liu**, **D.**, **Li**, **K.**, **Nepal**, **S.**, **Fu**, **A.**, and **Zou**, **Y.** (2022). Implicit Hammer: Cross-Privilege-Boundary Rowhammer through Implicit Accesses. In: *IEEE TDSC*.

[241] **Zheng**, **M.**, **Lou**, **Q.**, and **Jiang**, **L.** (2022). TrojViT: Trojan Insertion in Vision Transformers. arXiv:2208.13049.

[242] **Zhou**, **L.**, **Li**, **J.**, **Qiao**, **Z.**, **Ren**, **P.**, **Sun**, **Z.**, **Wang**, **J.**, **Wu**, **B.**, **Ji**, **Z.**, **Wang**, **R.**, **Cao**, **K.**, and **Huang**, **R.** (2023). Double-sided Row Hammer Effect in Sub-20 nm DRAM: Physical Mechanism, Key Features and Mitigation. In: *IRPS*.

[243] **Zhou**, **R.**, **Ahmed**, **S.**, **Rakin**, **A. S.**, and **Angizi**, **S.** (2023). DNN-Defender: An in-DRAM Deep Neural Network Defense Mechanism for Adversarial Weight Attack. arXiv:2305.08034.

[244] **Zhou**, **R.**, **Tabrizchi**, **S.**, **Roohi**, **A.**, and **Angizi**, **S.** (2022). LT-PIM: An LUT-Based Processing-in-DRAM Architecture With RowHammer Self-Tracking. In: *IEEE CAL*.

[245] **Zuravleff**, **W. K.** and **Robinson**, **T.** (1997). Controller for a Synchronous DRAM That Maximizes Throughput by Allowing Memory Requests and Commands to Be Issued Out of Order.