在不访问内存的情况下翻转其中的位：关于DRAM干扰错误的实验研究

Yoongu Kim[1] Ross Daly Jeremie Kim[1] Chris Fallin Ji Hye Lee[1] Donghyuk Lee[1]
Chris Wilkerson[2] Konrad Lai Onur Mutlu[1]

[1] 卡内基梅隆大学 [2] 英特尔实验室

摘要。内存隔离是可靠和安全计算系统的一个关键特性——对某个内存地址的访问不应对存储在其他地址的数据产生意外的副作用。然而,随着DRAM工艺技术缩小到更小的尺寸,防止DRAM单元之间的电相互作用变得更加困难。在本文中,我们揭示了商品DRAM芯片对干扰错误的脆弱性。通过从DRAM中的同一个地址读取数据,我们表明可以破坏附近地址的数据。更具体地说,在DRAM中激活同一行会破坏附近行的数据。我们使用一个恶意程序在英特尔和AMD系统上演示了这一现象,该程序产生了大量的DRAM访问。我们诱导了来自三个主要DRAM制造商的大多数DRAM模块(129个中的110个)出现错误。由此我们得出结论,许多已部署的系统可能面临风险。我们将干扰错误的根本原因确定为DRAM行的字线的反复切换,这会加剧单元间的耦合效应,加速附近行的电荷泄漏。我们使用基于FPGA的测试平台对干扰误差及其行为进行了广泛的特征研究。在我们的关键发现中,我们表明:(i) 仅需 13.9 万次访问就能引发一个误差;(ii) 多达每 1.7K 个单元中就有一个容易出错。在研究了各种潜在的解决问题的方法之后,我们提出了一种低开销的解决方案来防止这些误差。

1. 引言

动态随机存取存储器（DRAM）工艺技术的持续缩小使得更小的单元能够更紧密地排列在一起。在同一区域内塞进更多的DRAM单元具有众所周知的优势，即降低每位的存储成本。然而，增加单元密度对存储器的可靠性也有负面影响，原因有三。首先，小单元只能容纳有限的电荷量，这会降低其噪声容限，使其更容易出现数据丢失的情况[14, 47, 72]。其次，单元之间的近距离排列会引入电磁耦合效应，导致它们以不良的方式相互作用[14, 42, 47, 55]。第三，工艺技术的更大变化会增加异常容易受到单元间串扰影响的异常单元数量，加剧上述两种影响。

因此，高密度动态随机存取存储器（DRAM）更容易受到*干扰*，这是一种不同单元相互干扰的现象。如果一个单元受到的干扰超过其噪声容限，它就会发生故障并出现*干扰错误*。历史上，早在英特尔 1103 成为首款商业化的 DRAM 芯片时，DRAM 制造商就已经意识到了干扰错误[58]。为了缓解干扰误差，DRAM制造商一直在采用双管齐下的方法:*(i)*通过电路级技术改善单元间隔离[22, 32, 49, 61, 73]和*(ii)*在生产后测试中筛选出干扰误差[3, 4, 64]。我们证明，他们控制干扰误差的努力并不总是成功的，错误的DRAM芯片已经流入市场。

在本文中，我们揭示了当今市场上销售和使用的大容量 DRAM 芯片中存在干扰误差以及其广泛性。在我们分析的 129 个 DRAM 模块（包括 972 个 DRAM 芯片）中，我们发现 110 个模块（836 个芯片）存在干扰误差。特别是，过去两年（2012 年和 2013 年）生产的*所有*模块都容易受到干扰，这意味着在实际应用中出现干扰误差是相对较新的现象，影响的是更先进的工艺技术代。我们表明，对 DRAM 地址（更普遍地说，对 DRAM 行）进行少至 139K 次读取就会引发干扰误差。作为概念验证，我们构建了一个用户级程序，通过向相同地址发出多次加载操作，同时在中间刷新缓存行来持续访问 DRAM。我们证明，在英特尔或 AMD 机器上执行此类程序时，会引发许多干扰误差。

我们将动态随机存取存储器（DRAM）干扰错误的根本原因确定为被称为字线的内部线路上的电压波动。DRAM 由二维的单元阵列组成，其中每行单元都有自己的字线。要访问特定行中的单元，必须通过升高该行的字线电压来*激活它——即必须激活该行*。当同一行被多次激活时，这会迫使字线反复切换。根据我们的观察，这种字线电压波动对附近行产生了干扰效应，导致其中一些单元的电荷以加速的速度泄漏。如果这样的单元在恢复为原始值（即*刷新*）*之前损失了太多电荷*，它就会经历干扰错误。

我们在一个基于FPGA的测试平台上对DRAM干扰误差进行了全面特征分析，以了解其行为和症状。基于我们的研究结果，我们研究了多种潜在的解决方案（例如纠错和频繁刷新），但它们都存在一定的局限性。我们提出了一种有效且低开销的解决方案，称为*PARA*，它通过概率性地刷新那些可能面临风险的行来防止干扰误差。与其他解决方案相比，PARA不需要昂贵的硬件结构，也不会造成巨大的性能损失。本文做出了以下贡献。

Work done while at Carnegie Mellon University.

978-1-4799-4394-4/14/$31.00 c 2014 IEEE

1The industry has been aware of this problem since at least 2012, which is when a number of patent applications were filed by Intel regarding the problem of "row hammer" [6, 7, 8, 9, 23, 24]. Our paper was under review when the earliest of these patents was released to the public.

据我们所知，这是近年来第一篇揭示商品 DRAM 芯片中普遍存在干扰错误的论文。

我们构建了一个用户级程序，在真实系统（英特尔/AMD）上引入干扰误差。仅仅通过从动态随机存取存储器（DRAM）读取数据，我们就表明，这样的程序有可能突破内存保护，并损坏其不应被允许访问的页面中所存储的数据。

我们使用基于 FPGA 的测试平台和 129 个 DRAM 模块对 DRAM 干扰误差进行了广泛的表征。我们将干扰误差的根本原因确定为对一行字线的反复切换。我们观察到，由此产生的电压波动可能会干扰附近行的单元，导致它们以加快的速度失去电荷。在我们的关键发现中，我们表明:*(i)* 在 129 个模块中有 110 个存在可干扰单元；*(ii)* 多达 1.7K 个单元中的 1 个是可干扰的；*(iii)* 仅切换字线 139K 次就会导致干扰误差。

在研究了多种可能的解决方案后，我们提出了 PARA（概率*相邻行激活*），这是一种低开销的防止干扰误差的方法。每次切换字线时，PARA 以非常小的概率（$p < 1$）刷新附近行。随着字线被多次切换，干扰效应的增加被刷新附近行更高的可能性所抵消。

2. 动态随机存取存储器（DRAM）背景

在本节中，我们提供有关 DRAM 组织和操作的必要背景信息，以了解干扰错误的原因和症状。

2.1. 高级组织

DRAM芯片有多种配置[34]，目前容量范围为1-8 Gbit，数据总线宽度为4-16引脚。（特定的容量并不意味着特定的数据总线宽度。）单个DRAM芯片本身容量小，数据总线窄。这就是为什么通常会将多个DRAM芯片组合在一起，以提供大容量和宽数据总线（通常是64位）。这样的DRAM芯片组合被称为DRAM等*级*。一个或多个等级被焊接到电路板上，形成DRAM*模块*。

2.2. 低级组织

如图1a所示，DRAM由二维的DRAM*单元组成*，每个单元由一个*电容器*和一个*访问晶体管组成*。根据其电容器是充满电还是完全放电，一个单元*处于充电状态*或*放电*状态。这两种状态用于表示二进制数据值。

如图1b所示，每个单元位于两条垂直导线的交叉点：一条水平*字线*和一条垂直*位*线。字线在水平方向（行）上连接所有单元，位线在垂直方向（列）上连接所有单元。当一行字线被*提升*到高电压时，它使该行内的所有访问晶体管处于导*通*状态，从而将所有的电容器连接到各自的位线上。这使得该行的数据（以电荷的形式）能够转移到如图1a所示的*行缓冲器*中。*行缓冲器更常被称为读出放大器*，它从单元中读出电荷——这个过程会破坏数据。

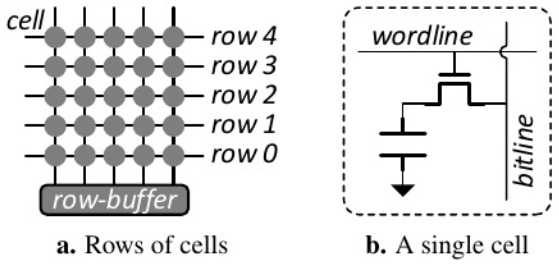

图 1 。动态随机存取存储器（DRAM）由单元组成

这些单元会立即将电荷写回单元中[38, 41, 43]。随后，对这一行的所有访问都由行缓冲器代表这一行提供服务。当不再对该行进行访问时，字线会*降低*到低电压，将电容器与位线断开连接。一组行被称为一个存储体，每个存储体都有自己的专用行缓冲器。（一个存储体的组织类似于图1a所示。）最后，多个存储体组合形成一个等级。例如，图2展示了一个2GB的等级，其256K行垂直划分为八个存储体，每个存储体有32K行，每行大小为8KB（D64Kb）[34]。拥有多个存储体可以增加并行性，因为对不同存储体的访问可以同时进行。

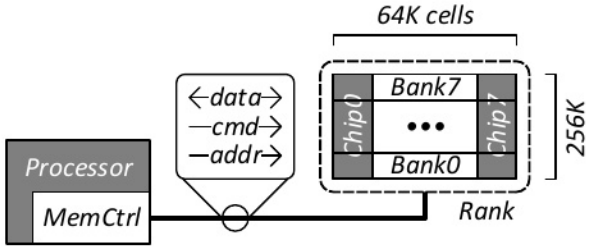

图 2 。内存控制器、总线、等级和存储体

2.3. 访问动态随机存取存储器（DRAM）

对内存中某一行数据的访问分三步进行:*(i)*在期望的存储体中“打开”期望的行；*(ii)*从行缓冲区访问期望的列；*(iii)*“关闭”该行。

1. 打开行。通过提升行的字线来打开一行。这会将该行连接到位线，并将其所有数据传输到存储器的行缓冲区。
2. 读取/写入列。行缓冲区的数据可通过根据需要读取或写入其任何列来进行访问。
3. 关闭行。在同一存储体中，在能够打开不同的行之前，必须通过降低其字线来关闭原始行。此外，行缓冲区会被清空。

*内存控制器*通常位于处理器中（图2），通过发出*命令*和*地址*来引导内存等级完成三个步骤，如表1所示。当一个内存等级接受一个命令后，需要一定的时间才能准备好接受另一个命令。这种延迟被称为DRAM*时序约束*[34]。例如，在同一行（在同一存储*体*）上的一对ACTIVATE命令之间的时序约束被称为t_{RC}（行周期时间），其典型值为50*纳秒*[34]。当试图尽可能快地打开和关闭同一行时，t_{RC}成为瓶颈——将最大速率限制为每t_{RC}一次。

操作	*命令*	*地址(们)*
1. 开启赛艇	激活（行动）	银行，行
2. “读/写专栏”	阅读/写作	银行，柱子
3. 关闭行	预充电（前置）	银行
刷新（第 2.4 节）	刷新（REF）	—

表 1 。动态随机存取存储器（DRAM）的命令和地址 [34]

2.4. 刷新动态随机存取存储器

动态随机存取存储器（DRAM)中存储的电荷不是持久的。这是由于各种泄漏机制，电荷可以通过这些机制分散：例如，亚阈值泄漏[56]和栅极诱导漏极泄漏[57]。最终，存储单元的电荷水平会偏离噪声容限，导致数据丢失——换句话说，一个存储单元的*保持*时间有限。在此时间到期之前，必须将存储单元的电荷恢复到其初始值：完全充电或完全放电。DDR3 DRAM 规范[34]保证至少 64 *毫秒*的保持时间，这意味着在此时间窗口内，一个等级内的所有存储单元都需要至少刷新一次。刷新一个存储单元可以通过打开其所属的行来实现。不仅行缓冲器读取存储单元的更改电荷值，同时还将其恢复为满值（第 2.2 节）。实际上，从电路的角度来看，*刷新一行和打开一行*是*相同*的*操作*。因此，内存控制器刷新一个等级的一种可能方式是连续向每一行发出 ACT 命令。在实践中，存在一个单独的REF命令，用于一次刷新多个行（表1）。当一个rank接收到REF命令时，它会通过内部生成ACT和PRE对来自动刷新最近刷新次数最少的几个行。在任何给定的64ms时间窗口内，内存控制器会发出足够数量的REF命令，以确保每个行都恰好被刷新一次。对于DDR3 DRAM rank，内存控制器在64ms内发出8192个REF命令，每7.8us发出一次（D64ms/8192）[34]。

3. 干扰误差的力学机制

一般来说，只要两个电路元件（例如电容器、晶体管、电线）之间存在足够强的相互作用，而它们本应相互隔离，就会产生干扰误差。根据哪个元件与其他元件相互作用以及它们如何相互作用，可能会有许多不同的干扰模式。

其中，我们发现了一种特定的干扰模式，它困扰着所有三大制造商的商品 DRAM 芯片。*当一条字线*的*电压反复切换时，附近行的一些单元会*以*快得多*的*速率泄漏电荷*。这些单元甚至无法保持电荷超过 64 毫秒，这是它们被刷新的时间间隔。最终，这导致单元丢失数据并出现干扰错误。

如果不从器件层面分析 DRAM 芯片，我们就无法就字线如何与相邻单元相互作用从而增加其泄漏性做出明确的说法。基于过去的研究和发现，我们假设可能存在三种相互作用方式。首先，改变字线的电压可能会通过.....向相邻字线注入噪声。

2At least one major DRAM manufacturer has confirmed these hypothe-ses as potential causes of disturbance errors.

电磁耦合[15, 49, 55]。这使得相邻行的访问晶体管在短时间内部分开启，促进电荷泄漏。其次，*桥*接是DRAM中一种众所周知的故障类型，其中在不相关的线和/或电容器之间形成了导电通道[3, 4]。一项关于嵌入式DRAM（eDRAM）的研究发现，切换字线可以加速两个桥接单元之间的电荷流动[29]。第三，据报道，在数百小时内切换字线可以通过*热载流子注入*永久性地损坏它[17]。如果一些热载流子注入到相邻的行中，这可能会改变它们单元中的电荷量或改变其访问晶体管的特性，从而增加其泄漏性。

只有在字线的累积干扰效应足够强大，足以扰乱附近单元的状态时，才会出现干扰误差。在下一节中，我们将展示一小段软件，它通过从 DRAM 中的同一行连续读取来实现这一点。

4. 实际系统演示

我们使用一个2GB的DDR3模块在英特尔（Sandy Bridge、Ivy Bridge和Haswell）和AMD（Piledriver）系统上诱导DRAM干扰错误。我们通过运行Code 1a来实现，这是一个程序，它在每次数据访问时都会对DRAM进行读取操作。首先，两个mov指令从地址X和Y的DRAM中读取数据，并将数据安装到寄存器和缓存中。其次，两个clflush指令将刚刚安装到缓存中的数据逐出。第三，mfence指令确保在执行任何后续内存指令之前，数据已完全刷新。最后，代码跳回到第一个指令，以进行另一次从DRAM读取的操作。（请注意，Code 1a不需要提升权限来执行其任何指令。）

```
1 code1a:
2   mov (X), %eax
3   mov (Y), %ebx
4   clflush (X)
5   clflush (Y)
6   mfence
7   jmp code1a
```

a. Induces errors

```
1 code1b:
2   mov (X), %eax
3   clflush (X)
4
5
6   mfence
7   jmp code1b
```

b. Does not induce errors

代码 1. 在英特尔/AMD 机器上执行的汇编代码

在无序处理器上，代码1a会生成多个DRAM读取请求，这些请求在发送到DRAM之前会先排队等待在内存控制器中：(req_X，req_Y，req_X，req_Y，）。重要的是，我们选择X和Y的值，使它们映射到*同*一个存储器银行，但映射到银行内的不同行。正如我们在第2.3节中所解释的，这迫使内存控制器反复打开和关闭这两个行：(ACT_X，$READ_X$，PRE_X，ACT_Y，$READ_Y$，PRE_Y，）。使用地址对（X，Y），我们随后执行了代码1a数百万次迭代。随后，我们重复了这一过程。

3Without the mfence instruction, there was a large number of hits in the processor’s *fill-buffer* [30] as shown by hardware performance counters [31].

4Whereas AMD discloses *which* bits of the physical address are used and *how* they are used to compute the DRAM bank address [5], Intel does not. We partially reverse-engineered the addressing scheme for Intel processors using a technique similar to prior work [46, 60] and determined that setting Y to XC8M achieves our goal for all four processors. We ran Code 1a within a customized Memtest86+ environment [1] to bypass address translation.

使用许多不同的地址对，直到2GB模块的每一行都被打开/关闭数百万次。最后，我们观察到Code 1a导致许多位翻转。对于每个处理器，表2报告了在模块的两种不同初始状态下（全0或全1），Code 1a引起的位翻转总数。由于Code 1a不向DRAM写入任何数据，我们得出结论，位翻转是干扰错误的体现。我们将在第6.1节中展示，这种特定的模块——我们称之为A_{19}（第5节）——在某些测试条件下会产生*数百万*个错误。

“位翻转”	桑迪桥	常春藤桥	哈斯韦尔	堆叠式击倒
“0” “1”	7;992	10 乘以 273	11;404	47
“1” “0”	8;125	10 乘以 449	11;467	12

表 2 。在 2GB 模块上由干扰引起的位翻转

作为对照实验，我们还运行了仅从单个地址读取数据的 Code 1b。正如我们所预期的，Code 1b 没有引发任何干扰错误。对于 Code 1b，它所有的读取操作都是针对 DRAM 中的同一行：(req_{X}、req_{X}、req_{X}、 ）。在这种情况下，内存控制器通过仅打开和关闭一次行来最小化 DRAM 命令的数量，同时在此期间发出许多列读取操作：(ACT_{X}、$READ_{X}$、$READ_{X}$、$READ_{X}$、 PRE_{X})。正如我们在第 3 节中所解释的，DRAM 干扰错误是由行的反复打开/关闭引起的，而*不是*列读取操作——这正是 Code 1b *不*引发任何错误的原因。

干扰误差会破坏内存应提供的两个不变量：*(i)* 读访问*不*应修改任何地址的数据；*(ii)* 写访问应*仅*修改被写入地址的数据。只要一行被反复打开，读和写访问都会引发干扰误差（第 6.2 节），所有这些误差都会出现在除被访问行之外的其他行中（第 6.3 节）。由于不同的 DRAM 行被映射（由内存控制器）到不同的软件页面[35]，代码 1a 仅仅通过访问其自身页面就可能损坏属于其他程序的页面。如果不加以控制，恶意程序可能会利用干扰误差来突破内存保护并危害系统。通过一些工程努力，我们相信我们能够将代码 1a 开发成一种*干扰攻击*，向其他程序注入错误、使系统崩溃，甚至可能劫持对系统的控制。我们将此类研究留给未来，因为这项工作的主要目标是理解和防止 DRAM 干扰误差。

5. 实验方法学

为了深入了解干扰误差，我们在一个基于 FPGA 的测试平台上对 129 个 DRAM 模块进行了特性描述。我们的测试平台使我们能够精确控制如何以及在逐周期基础上何时访问 DRAM。此外，它*不会*对写入 DRAM 的数据进行加密。[6]

测试平台。我们使用 DDR3-800 DRAM 内存控制器[71]、PCIe 2.0 核心[69]和定制的测试引擎对八块 Xilinx FPGA 板[70]进行了编程。在为每块 FPGA 板配备一个 DRAM 模块后，我们使用 PCIe 延长电缆将它们连接到两台主机计算机上。然后，我们将 FPGA 板与热电偶和加热器一起封闭在一个热室中，热电偶和加热器与外部温度控制器相连。除非另有说明，所有测试均在 50°C ± 0.2° C 的环境温度下运行。

测试。我们将*测试*定义为一系列专门设计为在模块中引发干扰错误的动态随机存取存储器（DRAM）访问。我们的大多数测试都源自上面列出的两段伪代码（代码2）：TestBulk和TestEach。TestBulk的目标是快速识别在多次切换每一行之后被干扰的所有单元的集合。另一方面，TestEach用于确定在多次切换每一行时哪些特定的单元被干扰。这两个测试都接受三个输入参数：*AI*（激活间隔）、*RI*（刷新间隔）和*DP*（数据模式）。首先，AI决定了行的切换频率——即执行一次内循环迭代所需的时间。其次，RI决定了在测试期间模块的刷新频率。最后，DP决定了在引入错误之前模块中填充的初始数据值。TestBulk（代码2a）首先将DP写入整个模块。然后，以AI的频率切换一行，持续RI的时间——即行被切换N D .2 *RI/=AI次*。然后，对模块中的每一行重复此过程。最后，TestBulk 会读出整个模块并识别出所有受干扰的单元。TestEach（代码 2b）类似，只是将第 6 行、第 12 行和第 13 行移到了外部 for 循环内部。仅切换一行后，TestEach 会读出模块并识别出受该行干扰的单元。

```
1
```

```
测试（人工智能；机器人智能;数据处理）
2     设置人工智能（AI）
3     设置相对指数（RI）
4     N      .2 电阻抗成像/等电抗成像
5
6     写入所有（DP）
7     对于 r      0   最大行数 0 N 而言
8         因为我是第 i 行第
9             ACT 列
              读取第 0 列。第
10            r 行的前一个元素
11
12    “readAll()”
13    “查找错误”
```

a. 一次性测试所有行

```
1  测试每个（人工智能；机器人智能;数据处理）
2     设置人工智能（AI）
3     设置相对指数（RI）
4     N      .2 电阻抗成像/等电抗成像
5
6     对于 r      0   最大行数 而言
7         写入所有（DP）
8         对于 i 0 N 实际行
              动在第行
9             读取第 0 列。第
10            r 行的前一个元素
11
12        “readAll()”
13        “查找错误”
```

b. 一次测试一行

代码 2. 在 FPGA 上合成的两种类型的测试

测试参数。在我们的大多数测试中，我们将 AI 设置为 55 纳秒，RI 设置为 64 毫秒，对应的 N 值为 2:33 10^（-6）。我们选择 55 纳秒作为 AI 是因为它接近在不违反 t_{RC} 时序约束的情况下翻转一行的最大速率（第 2.3 节）。在某些测试中，我们还把 AI 扫频到 500 纳秒。我们选择 64 毫秒作为 RI，因为这是 DDR3 DRAM 标准指定的默认刷新间隔（第 2.4 节）。在某些测试中，我们还把 RI 扫频到 10 毫秒和 128 毫秒。对于 DP，我们主要使用两种数据模式[65]：

5The faster a processor accesses DRAM, the more bit-flips it has. Ex-pressed in the unit of accesses-per-second, the four processors access DRAM at the following rates: 11.6M, 11.7M, 12.3M, and 6.1M. (It is possible that not all accesses open/close a row.)

6We initialize the module by making the processor write out all ‘0’s or all ‘1’s to memory. But before this data is actually sent to the module, it is *scrambled* by the memory controller to avoid electrical resonance on the DRAM data-bus [31]. In other words, we do not know the exact “data” that is received by the module. We examine the significance of this in Section 6.4.

不同行的刷新间隔彼此不对齐（第 2.4 节）。因此，我们将一行切换*两次*，持续时间为 *RI* 的两倍，以确保与该行的至少一个刷新间隔完全重叠。

制造商模块		日期	Timing☒		组织		芯片			每模块的受害者数量			RIth(毫秒)
		(yy - ww)频率(兆赫/秒)tRC(纳秒)大小(GB)芯片尺寸					(Gb)☒	别针	DieVersion☒	平均	最小 最大		分钟
A 总计 43 个模块	A1	10-08	1066	50.625	0.5	4	1	16	B	0	0	0	–
	A2	10-20	1066	50.625	1	8	1	8	F	0	0	0	–
	A3 - 5	10-20	1066	50.625	0.5	4	1	16	B	0	0	0	–
	A6 - 7	11-24	1066	49.125	1	4	2	16	D	7:8 10 - 1	5:2 10 - 1	1:0 10^(-2)	21:3
	A8 - 12	11-26	1066	49.125	1	4	2	16	D	2:4 10 - 2	5:4 10 - 1	4:4 10^(-2)	16:4
	A13 - 14	11-50	1066	49.125	1	4	2	16	D	8:8 10 - 1	1:7 10 - 1	1:6 10^(-2)	26:2
	A15 - 16	12-22	1600	50.625	1	4	2	16	D	9:5	9	1:0 10 - 1	34:4
	A17 - 18	12-26	1600	49.125	2	8	2	8	M	1:2 10 - 2	3:7 10 - 1	2:0 10^(-2)	21:3
	A19 - 30	12-40	1600	48.125	2	8	2	8	K	8:6 10 - 6	7:0 10^(-6)	1:0 10^(-7)	8:2
	A31 - 34	13-02	1600	48.125	2	8	2	8	–	1:8 10^(-6)	1:0 10^(-6)	3:5 10^(-6)	11:5
	A35 - 36	13-14	1600	48.125	2	8	2	8	–	4:0 10 - 1	1:9 10 - 1	6:1 10 - 1	21:3
	A37 - 38	13-20	1600	48.125	2	8	2	8	K	1:7 10^(-6)	1:4 10^(-6)	2:0 10^(-6)	9:8
	A39 - 40	13-28	1600	48.125	2	8	2	8	K	5:7 10^(-4)	5:4 10^(-4)	6:0 10^(-4)	16:4
	A41	14-04	1600	49.125	2	8	2	8	–	2:7 10^(-5)	2:7 10^(-5)	2:7 10^(-5)	18:0
	A42 - 43	14-04	1600	48.125	2	8	2	8	K	0:5	0	1	62:3
B 总计 54 个模块	B1	08-49	1066	50.625	1	8	1	8	D	0	0	0	–
	B2	09-49	1066	50.625	1	8	1	8	E	0	0	0	–
	B3	10-19	1066	50.625	1	8	1	8	F	0	0	0	–
	B4	10-31	1333	49.125	2	8	2	8	C	0	0	0	–
	B5	11-13	1333	49.125	2	8	2	8	C	0	0	0	–
	B6	11-16	1066	50.625	1	8	1	8	F	0	0	0	–
	B7	11-19	1066	50.625	1	8	1	8	F	0	0	0	–
	B8	11-25	1333	49.125	2	8	2	8	C	0	0	0	–
	B9	11-37	1333	49.125	2	8	2	8	D	1:9 10^(-6)	1:9 10^(-6)	1:9 10^(-6)	11:5
	B10 - 12	11-46	1333	49.125	2	8	2	8	D	2:2 10^(-6)	1:5 10^(-6)	2:7 10^(-6)	11:5
	B13	11-49	1333	49.125	2	8	2	8	C	0	0	0	–
	B14	12-01	1866	47.125	2	8	2	8	D	9:1 10 - 5	9:1 10 - 5	9:1 10 - 5	9:8
	B15 - 31	12-10	1866	47.125	2	8	2	8	D	9:8 10 - 5	7:8 10^(-5)	1:2 10^(-6)	11:5
	B32	12-25	1600	48.125	2	8	2	8	E	7:4 10 - 5	7:4 10 - 5	7:4 10 - 5	11:5
	B33 - 42	12-28	1600	48.125	2	8	2	8	E	5:2 10^(-5)	1:9 10^(-5)	7:3 10^(-5)	11:5
	B43 - 47	12-31	1600	48.125	2	8	2	8	E	4:0 10^(-5)	2:9 10^(-5)	5:5 10 - 5	13:1
	B48 - 51	13-19	1600	48.125	2	8	2	8	E	1:1 10^(-5)	7:4 10^(-4)	1:4 10^(-5)	14:7
	B52 - 53	13-40	1333	49.125	2	8	2	8	D	2:6 10^(-4)	2:3 10^(-4)	2:9 10^(-4)	21:3
	B54	14-07	1333	49.125	2	8	2	8	D	7:5 10^(-3)	7:5 10^(-3)	7:5 10^(-3)	26:2
C 总计 32 个模块	C1	10-18	1333	49.125	2	8	2	8	A	0	0	0	–
	C2	10-20	1066	50.625	2	8	2	8	A	0	0	0	–
	C3	10-22	1066	50.625	2	8	2	8	A	0	0	0	–
	C4-5	10-26	1333	49.125	2	8	2	8	B	8:9 10 - 2	6:0 10^(-2)	1:2 10^(-3)	29:5
	C6	10-43	1333	49.125	1	8	1	8	T	0	0	0	–
	C7	10-51	1333	49.125	2	8	2	8	B	4:0 10^(-2)	4:0 10^(-2)	4:0 10^(-2)	29:5
	C8	11-12	1333	46.25	2	8	2	8	B	6:9 10 - 2	6:9 10 - 2	6:9 10 - 2	21:3
	C9	11-19	1333	46.25	2	8	2	8	B	9:2 10 - 2	9:2 10 - 2	9:2 10 - 2	27:9
	C10	11-31	1333	49.125	2	8	2	8	B	3	3	3	39:3
	C11	11-42	1333	49.125	2	8	2	8	B	1:6 10^(-2)	1:6 10^(-2)	1:6 10^(-2)	39:3
	C12	11-48	1600	48.125	2	8	2	8	C	7:1 10^(-4)	7:1 10^(-4)	7:1 10^(-4)	19:7
	C13	12-08	1333	49.125	2	8	2	8	C	3:9 10^(-4)	3:9 10^(-4)	3:9 10^(-4)	21:3
	C14-15	12-12	1333	49.125	2	8	2	8	C	3:7 10^(-4)	2:1 10^(-4)	5:4 10^(-4)	21:3
	C16-18	12-20	1600	48.125	2	8	2	8	C	3:5 10^(-3)	1:2 10^(-3)	7:0 10^(-3)	27:9
	C19	12-23	1600	48.125	2	8	2	8	E	1:4 10^(-5)	1:4 10^(-5)	1:4 10^(-5)	18:0
	C20	12-24	1600	48.125	2	8	2	8	C	6:5 10^(-4)	6:5 10^(-4)	6:5 10^(-4)	21:3
	C21	12-26	1600	48.125	2	8	2	8	C	2:3 10^(-4)	2:3 10^(-4)	2:3 10^(-4)	24:6
	C22	12-32	1600	48.125	2	8	2	8	C	1:7 10^(-4)	1:7 10^(-4)	1:7 10^(-4)	22:9
	C23 - 24	12-37	1600	48.125	2	8	2	8	C	2:3 10^(-4)	1:1 10^(-4)	3:4 10^(-4)	18:0
	C25-30	12-41	1600	48.125	2	8	2	8	C	2:0 10^(-4)	1:1 10^(-4)	3:2 10^(-4)	19:7
	C31	13-11	1600	48.125	2	8	2	8	C	3:3 10^(-5)	3:3 10^(-5)	3:3 10^(-5)	14:7
	C32	13-35	1600	48.125	2	8	2	8	C	3:7 10^(-4)	3:7 10^(-4)	3:7 10^(-4)	21:3

我们报告芯片封装上标记的生产日期，这比从模块中获取的其他日期更准确。☒我们报告存储在模块板载只读存储器（ROM）中的时序约束[33]，系统 BIOS 会读取该约束来校准内存控制器。

我们的测试平台所支持的动态随机存取存储器（DRAM）芯片的最大尺寸为 2 千兆字节。

我们报告了芯片封装上标记的动态随机存取存储器（DRAM）芯片版本，其通常以以下方式发展：M A B C 。

表 3. 129 个 DDR3 动态随机存取存储器模块的样本群体，按制造商分类，并按生产日期排序

行条纹（偶数/奇数行填充“0”/“1”）及其逆行条纹。正如第 6.4 节将展示的，这两种数据模式引起的错误最多。在一些测试中，我们还使用了纯*色*、*列条纹*、*方格*以及它们的逆变体[65]。

DRAM 模块。如表 3 中所列，我们对总共 129 个 DDR3 DRAM 模块进行了干扰误差测试。它们包括来自三家制造商的 972 个 DRAM 芯片，这三家制造商的名称已被匿名化为 A、B 和 C。8 这三家制造商在全球 DRAM 市场中占据了很大份额[20]。我们使用以下符号来指代这些模块：M_i

（M 代表制造商，i 代表数字标识符，*yyww* 代表制造日期的年份和周数）。9 就制造商、制造日期和芯片类型而言，有些模块彼此难以区分（例如，$A_{3\text{-}5}$）。我们将这样一组模块统称为一个系列。对于多级模块，仅第一级在表 3 中有所体现，这也是我们测试的唯一一级。我们将交替使用“模块”和“级”这两个术语。

6. 表征结果

我们现在呈现我们的特性研究结果。第 6.1 节解释了模块中的干扰误差数量如何因制造商和制造日期的不同而有很大差异。第 6.2 节证实，反复激活一行确实是干扰误差的来源。此外，我们还测量了在误差开始出现之前一行必须被激活的最小次数。第 6.3 节表明，由这样一行（即*攻击行*）引起的误差主要局限于另外两行（即*受害行*）。然后我们提供了关于受害行很可能是紧邻行的理由。第 6.4 节表明，干扰误差只影响已充电的单元，导致它们因放电而失去数据。

6.1. 干扰误差很普遍

对于表 3 中的每个模块，我们试图通过对其执行两次 TestBulk 测试来引入干扰误差：

1. 测试（55 纳秒，64 毫秒，行条纹）
2. 测试（55 纳秒，64 毫秒，行条纹）

如果一个单元在任何一次运行中遇到错误，我们就称其为该模块的*受害单元*。有趣的是，在任何模块中，实际上没有单元在两次运行中同时出错——这意味着两次运行中错误的总和等于一个模块的独特受害单元的数量。10 （这是一个重要的观察结果，将在第 6.4 节中进一步探讨。）

对于每个模块家族，表3中的三个右侧列报告了属于该家族的模块中受害者的平均/最小/最大数量。如表所示，除了19个模块外，我们能够对所有模块引入错误，其中大多数也是每个制造商最旧的模块。事实上，存在一些日期界限，将有错误的模块与无错误的模块区分开来。对于A、B和C，它们各自的日期界限是 2011 - 24、2011 - 37 和 2010 - 26。除了 A_{42}、B_{13} 和 C_6 之外，在这些日期之后生产的每个模块都存在误差。这些日期界限很可能表明工艺升级，因为它们也与晶圆版本升级相重合。以制造商 B 为例，在界限之前的 2Gb 8 芯片的晶圆版本为 C，而界限之后（除了 B_{13}）之外）的芯片的晶圆版本为 D 或 E。因此，我们得出结论，干扰误差是相对较新的现象，几乎影响了过去三年内生产的几乎所有模块。

利用表3的数据，图3绘制了每类模块的归一化错误数与制造日期的关系。误差棒表示每类模块的最小和最大值。从图中可以看出，2012年至2013年的模块特别容易出错。对于每个制造商，每10-9个单元的受害者数量可以达到5:9 10^{-5}，1:5 10^{-5}，和1:9 10^{-4}。有趣的是，图3揭示了一种类似拼图一样的趋势，即错误的数量突然增加后逐渐下降。这可能发生在制造商从旧但可靠的工艺迁移到新但不可靠的工艺时。随着时间的推移，通过不断调整，新工艺最终可能会再次变得可靠——这可以解释为什么制造商A ($A_{42\text{-}43}$)的最新模块几乎没有错误。

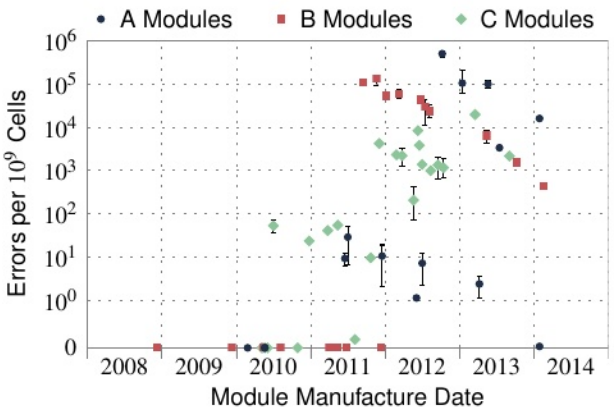

图 3 。归一化错误数量与制造日期的关系

6.2. 访问模式依赖

到目前为止，我们通过反复打开、读取和关闭同一行来展示干扰误差。我们使用以下符号来表示这种访问模式，其中N是一个大数：（打开-读取-关闭）N。然而，这并不是引发错误的唯一访问模式。表4列出了总共四种不同的访问模式，其中两种在我们测试的模块上引发了错误：A_{23}、B_{11}和C_{19}。选择这三个模块是因为它们在所有来自同一制造商的模块中具有最多的错误（A_{23}和B_{11})或第二多的错误(C_{19})）。前两种访问模式的共同点是它们反复打开和关闭同一行。相比之下，另外两种模式只执行一次，并且没有引发任何错误。由此我们得出结论，对同一字线的反复切换确实是干扰误差的原因。

8We tried to avoid third-party modules since they sometimes obfuscate the modules, making it difficult to determine the actual chip manufacturer or the exact manufacture date. Modules $B_{14\text{-}31}$ are engineering samples.

9Manufacturers do not explicitly provide the technology node of the chips. Instead, we interpret recent manufacture dates and higher die versions as rough indications of more advanced process technology.

10In some of the B modules, there were some rare victim cells (15) that had errors in both runs. We will revisit these cells in Section 6.3.

11For write accesses, a row cannot be opened and closed once every t_{RC} due to an extra timing constraint called t_{WR} (write recovery time) [34]. As a result, the second access pattern in Table 4 induces fewer errors.

访问模式	干扰误差?
1. （打开 - 读取 - 关闭）名词	是
2. （开放 - 写入 - 关闭）名词	是
3. *打开 - 读取*^{N} - 关闭	否
4. *开放写入*^{N} - 关闭	否

表 4 。引发干扰误差的访问模式

刷新间隔（RI）。如第5节所述，我们的测试每55纳秒打开一行。对于每一行，我们在整个RI期间保持这个速率（默认值为64毫秒）。这是为了让这一行能够最大限度地干扰其他单元，导致它们在下次刷新前泄漏最多的电荷。当RI在10-128毫秒之间变化时，图4显示了三个模块中的错误数量。由于时间限制，我们只测试了第一个模块。对于较短的RI，由于两个原因，错误数量较少：(*i*)受害单元在刷新之间泄漏电荷的时间更少；*(ii)*在这些刷新之间，一行打开的次数更少，从而减少了对受害单元的干扰效果。在足够短的RI下——我们称之为*阈值刷新间隔*RI_{th}——错误不仅在第一模块中完全消除，而且在整个模块中也是如此。对于每个模块家族，表3的最右侧列报告了该家族中属于该家族的模块的最小⊠。在RI为64毫秒时，受害单元最多的家族也可能具有最低的RI_{th}: 8.2毫秒、9.8毫秒和14.7毫秒。这意味着刷新频率分别增加了 7.8 倍、6.5 倍和 4.3 倍。

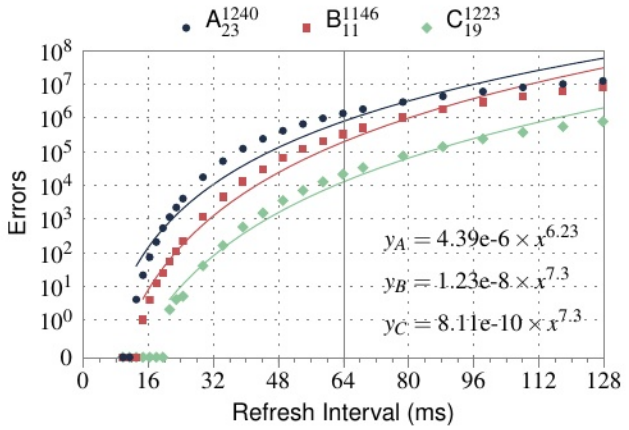

图 4 。随着刷新间隔的变化的错误数量

激活间隔（AI）。当 AI 在 55 - 500 纳秒之间变化时，图 5 绘制了三个模块中的错误数量。（仅测试第一组，并且读取间隔（RI）恒定在 64 毫秒。）对于较长的 AI，错误较少，因为一行打开的频率降低，从而减少了其干扰效应。当 AI 足够长时，三个模块都没有错误：500 纳秒、450 纳秒和 250 纳秒。然而，在最短的 AI 下，趋势出现了显著逆转：B_{11}和 C_{19}在 60 纳秒时的错误数量比在 65 纳秒时少。当一行打开的频率更高时，错误数量怎么会更少呢？只有在 60 纳秒时打开一行的干扰效应比 65 纳秒时更弱的情况下，这种异常现象才能解释得通。一般来说，如果在打开一行时字线电压没有迅速升高，则已知行耦合效应会减弱[55]。字线电压则由称为字线*电荷泵*的电路升高，如果该电路在执行任务后没有给够时间“恢复”，就会变得迟缓。12 当每 60 纳秒提升一次字线时，我们假设电荷泵在每个间隔结束时无法完全恢复其全部能力，这会导致字线上的电压转换缓慢，最终产生较弱的干扰效应。相比之下，55 纳秒的 AI 似乎不受这种现象的影响，因为错误数量出现了大幅增加。我们认为这是我们的内存控制器安排刷新命令的方式所导致的一种现象。在 55 纳秒时，我们的内存控制器碰巧以 100%的利用率运行，这意味着其缓冲区中总是有一个 DRAM 请求排队等待。为了尽量减少请求的延迟，内存控制器将待定的刷新命令的优先级降低 64 微秒。这种技术完全符合 DDR3 DRAM 标准[34]，并且在通用处理器中广泛应用[31]。结果，有效的刷新间隔略有延长。

ened，这再次增加了错误的数量。

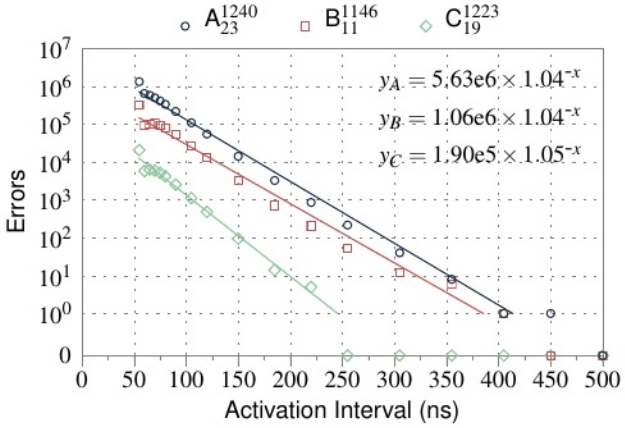

图 5.随着激活间隔的变化，错误的数量

激活次数。我们已经看到，干扰误差受到 RI 和 AI 长度的严重影响。在图 6 中，我们将前两个图叠加在一起，比较它们的影响。两个图都已归一化到相同的 x 轴，其值对应于每个刷新间隔的激活次数：RI = AI.13（对于图 4，仅显示左半部分，其中 RI 64ms）。在图 6 中，当 RI 和 AI 设置为默认长度时，激活次数达到最大值 1:14 10^{-6}（D64ms/55ns）。在这个特定点，两项研究之间的误差数量退化到相同的值。从图中可以清楚地看出，激活次数越少，误差越少。对于相同的激活次数，长 RI 和长 AI 比短 RI 和短 AI 更可能引发更多误差。我们将*阈值激活次数*定义为当 RI 为 64ms 时引发误差所需的最小激活次数。三个模块（仅针对其第一个存储体）的阈值激活次数分别为 139K、155K 和 284K。

12The charge-pump “up-converts” the DRAM chip’s supply voltage into an even higher voltage to ensure that the wordline’s access-transistors are completely switched on. A charge-pump is essentially a large reservoir of charge which is slowly refilled after being tapped into.

13The actual formula we used is (*RI* 8192 t_{RFC}/=*AI*, where t_{RFC} (refresh cycle time) is the timing constraint between a REF and a subsequent ACT to the same module [34]. Our testing platform sets t_{RFC} to 160*ns*, which is a sufficient amount of time for all of our modules.

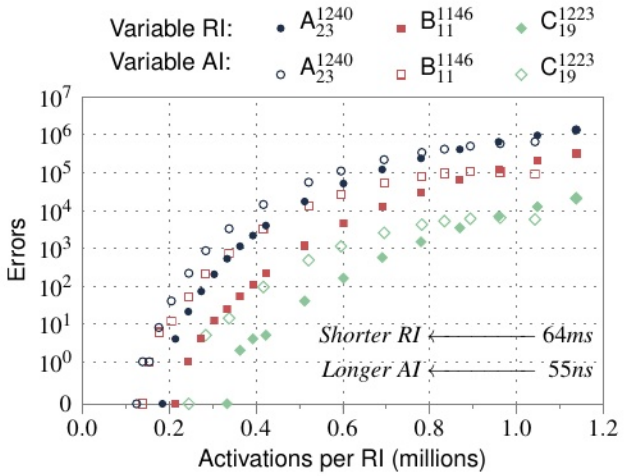

图 6 。错误数量与激活次数

6.3. 地址关联：攻击者与受害者

在 A_{23}、B_{11} 和 C_{19} 中，大多数行至少有一个单元格出现错误：分别为 100%、99.96%和 41.54%。我们分析了这些受害单元格的地址，以确定它们是否表现出任何空间局部性。我们未能识别到任何明显的模式或偏差。然而，偶然情况下，一些受害单元格最终仍可能彼此靠近。对于这三个模块，表 5 展示了在其完整地址空间（0 - 2GB）中有多少 64 位字包含 1 个、2 个、3 个或 4 个受害单元格。虽然大多数字仅有一个受害单元格，但也存在一些字有多个受害单元格。这对纠错码（ECC）有重要影响。例如，SECDED（单纠错，双差错检测）只能纠正一个 64 位字内的单位错误。然而，如果一个字包含两个受害单元格，SECDED 就无法纠正由此产生的双位错误。对于三个或更多的受害单元格，SECDED 甚至无法检测到多位错误，导致数据损坏无声发生。因此，我们得出结论，SECDED 对干扰错误并非万无一失。

模块	*具有 X 个错误的 64 位字的数量*			
	*X*D1	*X*D2	*X*D3	*X*D4
A_{23}	9；709；721	181;856	2;248	18
B_{11}	2；632；280	13;638	47	0
C_{19}	141;821	42	0	0

表 5 。不可纠正的多位错误（加粗）

在 A_{23}、B_{11} 和 C_{19} 中，大多数行在反复打开时会导致错误。我们将此类行称为*攻击*行。我们通过仅对第一个存储库运行两次 TestEach 来暴露模块中的攻击行：

1. 测试每个（55 纳秒，64 毫秒，行条纹）
2. 测试每个（55 纳秒，64 毫秒，行条纹）

这三个模块的攻击者行数分别为：32768 行、32754 行和 15414 行。考虑到模块中一个存储库有 32K 行，我们得出结论，很大一部分行都是攻击者：分别为 100%、99.96%和 47.04%。

每个攻击行都可以与一组在两次测试中任一次受到攻击者干扰的受害细胞相关联。图7展示了这三个模块中这一组细胞的大小分布。

A_{23}中的攻击行是最强大的，一次能干扰多达110个细胞。（我们无法解释图中的两个峰值。）另一方面，B_{11}和C_{19}中的攻击者分别能干扰多达28和5个细胞。

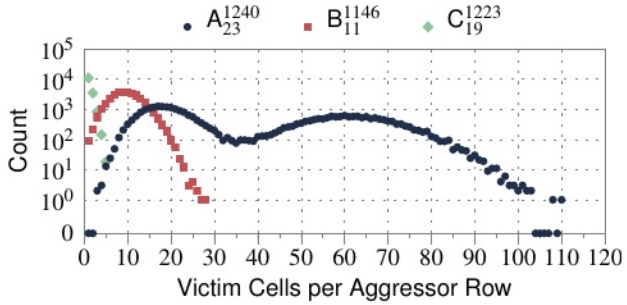

图 7. 侵略者一行影响了多少个单元格?

同样，我们可以将每个攻击者行与一组*受害者行*相关联，受害者单元格就属于这些受害者行。图8绘制了这一组的大小分布。我们看到，攻击者行的受害者单元格主要集中在两行或更少。事实上，只有一小部分攻击者行影响三行或更多：2.53%、0.0122%和0.00649%。

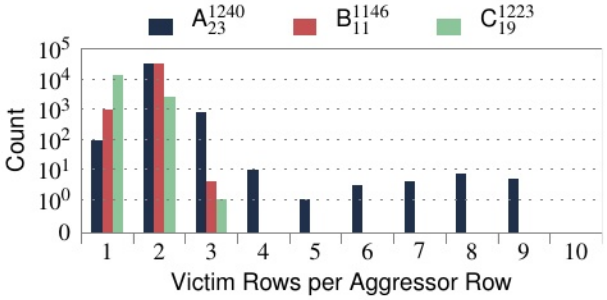

图 8 。一个攻击行会影响多少行?

为了确定攻击行地址与其受害者行地址之间是否存在任何相关性，我们形成了它们之间的所有可能对。对于每一对这样的对，我们随后计算行地址差值如下：*受害者行*地址 *攻击行地址*:这些差值的直方图如图 9 所示。从图中可以清楚地看出，攻击行只会导致除自身以外的行出现错误。这是可以理解的，因为每次打开和关闭攻击行时，它还有助于补充自身所有单元中的电荷（第 2.4 节）。由于攻击行的单元在不断被刷新，它们极不可能泄漏足够的电荷而导致数据丢失。

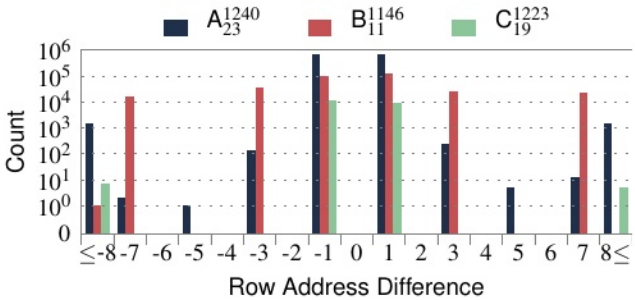

图 9. 哪些行会受到攻击行行的影响?

对于这三个模块，图9显示在1处有强烈的峰值，表明攻击者和其受害者很可能具有连续的行地址，即它们在*逻辑*上是相邻的。然而，逻辑上相邻并不总是意味着它们在硅片上物理上相邻，即物理*上相邻*。尽管每个逻辑行都必须映射到某个物理行，但它们如何映射完全取决于DRAM制造商的决定[65]。尽管如此，我们假设攻击者会导致其物理相邻行出现错误，原因有三。

原因 1. 字线电压波动很可能会给紧邻的行施加最大的电应力[49, 55]。

原因 2. 根据定义，一行只有两个紧邻的邻行，这或许可以解释为什么干扰误差主要局限于两行。

原因 3. 逻辑上的相邻性可能与物理上的相邻性高度相关，我们从图 9 中 1 处的强峰值可以推断出这一点。

然而，我们在图 8 和图 9 中也看到了差异，其中攻击行似乎会导致非相邻行的错误。我们假设这是由于两个原因。

原因 1. 在图 8 中，一些攻击者影响的不仅仅是两行。这可能是由*重新*映射的行导致的不规则性。回到图 2（第 2.1 节），一个等级的第 i 个“行”是通过在每个芯片中选取*第 i 行*并将它们连接起来形成的。但如果其中一个芯片中的行有故障，制造商会将其重新映射到备用行（例如，i j）[28]。在这种情况下，第 i 个“行”有*四*个紧邻的邻居：七个芯片中的第 i ± 1 *行*以及重新映射芯片中的第 j ± 1 行。

原因 2. 在图 9 中，一些侵略者影响的行并非逻辑上相邻的：例如，在 3 和 7 处的侧峰。这可能是制造商相关映射的产物，其中一些物理上相邻的行具有逻辑行地址，相差 3 或 7 -例如，当地址是格雷编码时[65]。或者，也有可能是侵略者影响的行比紧邻的行距离更远——这种可能性我们无法完全排除。然而，如果是这种情况，那么这些峰值不太可能以 2、4 和 6 的间隔分开。14

双重攻击者行。大多数受害单元仅受到*单个*攻击者行的干扰。然而，有一些受害单元会受到*两个不同*攻击者行的干扰。在三模块的第一组中，此类受害单元的数量分别为 83、2 和 0。例如，在模块 A_{23}中，位于（第 1464 行，第 50466 列）的受害单元在 1463 行*或* 1465 行被切换时会出现“1”到“0”的错误。在模块 B_{11}中，位于（第 5907 行，第 32087 列）的受害单元在 5906 行被切换时会出现“0”到“1”的错误，而在 5908 行被切换时会出现“1”到“0”的错误。在这两个模块中，分别对于具有两个攻击者行的其他受害单元，*也存*在同样的趋势。有趣的是，这两个在具有两个攻击者行的 B_{11} 模块中的受害细胞也是我们在第 6.1 节所述的测试对的两轮运行中均出现错误的相同细胞。这些细胞是我们观察到的在同一细胞中同时出现“0”→“1”*和*“1”→“0”错误的唯一情况。除了仅在 B 模块中发现的此类罕见异常情况外，由于我们接下来要解释的原因，其他所有受害细胞仅在单个优先方向出现错误。

6.4. 数据模式依赖

到目前为止，我们对所有错误一视同仁，没有对错误的两个不同*方向*（“0”到“1”和“1”到“0”）进行任何区分。当我们根据错误的方向在表3中对其进行分类时，一个有趣的趋势显现出来。A模块对两个方向没有偏好，而B和C模块则严重偏向“1”到“0”的错误。平均到每个模块，A、B和C的“1”到“0”错误的相对比例分别为49.9%、92.8%和97.1%。

干扰误差看似不对称的特性*与*动态随机存取存储器（DRAM）单元的一个内在特性有关，称为*方向性*。根据实现方式，一些单元使用充电状态表示逻辑值“1”，而其他单元则使用放电状态表示逻辑值“1”——这些单元分别被称为*真单元和反*单元[44]。如果真单元失去电荷，就会发生“1”到“0”的错误。当我们对两个模块（B_{11}和C_{19}）进行性能分析时，发现它们主要由真单元组成，比例约为1000:1。对于这两个模块，真单元的主导地位及其“1”到“0”错误意味着受害单元在受到干扰时最有可能*失去*电荷。对于A_{23}模块，其地址空间被划分为交替的大片真单元和反单元区域，每512行交替一次。对于这个模块，我们发现，在真单元占主导地位的行（0-511、1024-1535、2048-2559等）中，“1”到“0”错误占主导地位（>99.8%）。相比之下，在反单元占主导地位的其余行中，“0”到“1”错误占主导地位（>99.7%）。无论其方向如何，一个单元只有在初始充电的情况下才会*失去电荷*——这就解释了为什么在 6.1 节的测试中，给定的单元在两次运行中都没有出现错误。由于这两次运行用反向数据模式填充模块，一个单元不可能在这两次运行中都充电。

表6报告了使用四种不同的数据模式及其反转模式（Solid、RowStripe、ColStripe和Checkered）在三个模块中引起的错误数量。其中，RowStripe（偶数/奇数行中‘0‘/'1'）在A_{23}和B_{11}中引起的错误最多，在C_{19}中引起的错误次之。相比之下，Solid（全’0’）在所有三个模块中引起的错误最少，相差一个数量级甚至更多。如果对干扰错误的要求仅仅是两个方面：*(i)*受害单元处于充电状态，*(ii)*其攻击行被切换，那么这种巨大的差异就*无法*解释。这是因为所有四种数据模式都满足这两个要求。相反，除了受害单元与攻击字线的耦合外，一定还有其他因素在起作用。事实上，我们发现大多数

14Figure 9 presents further indications of re-mapping, where some mod-ules have non-zero values for ‣8 or beyond. Such large differences —which in some cases reach into the thousands —may be caused when a faulty row is re-mapped to a spare row that is far away, which is typically the case [28].

15For manufacturer C, we excluded modules with a die version of B. Unlike other modules from the same manufacturer, these modules had errors that were evenly split between the two directions.

16At 70☒C, we wrote all ‘0’s to the module, disabled refreshes for six hours and read out the module. We then repeated the procedure with all ‘1’s. A cell was deemed to be true (or anti) if its outcome was ‘0’ (or ‘1’) for both experiments. We could not resolve the orientation of every cell.

受害细胞与存储在其他一些细胞中的数据相关联。17 受害细胞可能有*攻击细胞(通常位于攻击行)*，这些攻击细胞必须被释放，受害细胞才会出错。受害细胞也可能有*保护细胞(通常位于攻击行或受害行)*，这些保护细胞必须被充电或释放，受害细胞才会出错的概率降低。一般来说，干扰误差似乎是一种复杂的“N 体”现象，涉及多个细胞的相互作用，其最终结果只能解释表 6 中的差异。

模块	测试批量（DP） C 测试批量（DP）			
	坚固	行条纹	彩色条纹	格子状
A_{23}	112；123	1；318；603	763；763	934；536
B_{11}	12;050	320;095	9 乘以 610	302；306
C_{19}	57	20;770	130	29；283

表 6. 不同数据模式的错误数量

7. 灵敏度结果

错误大多是可重复的。我们对三个模块进行了十*次迭代*测试，每次迭代都包括第6.1节中描述的测试对。在十次迭代中，仅第一个模块的平均错误数分别为：1.31M、339K和21.0K。所有三个模块的每次迭代中，没有一次偏离平均数超过0.25%。十次迭代中，独特受害细胞的总数分别为：1.48M、392K和24.4K。大多数受害细胞都是重复犯错的，这意味着它们在每次迭代中都有错误：78.3%、74.4%和73.2%。然而，有些受害细胞只在一次迭代中出现错误：3.14%、4.86%和4.76%。这意味着，要彻底搜索所有可能的受害细胞，需要进行大量的迭代，需要连续测试几天（或更长时间）。减少测试时间的一种可能方法是，像我们在图4（第6.2节）中所做的那样，将RI增加到标准值64ms以上。然而，由于在RI128ms时的一次迭代并不能100%覆盖所有在RI64ms时的受害细胞，因此可能还是需要多次迭代：99.77%、99.87%和99.90%。

受害单元 ¤ 弱单元。尽管每个 DRAM 单元的保持时间需要大于 64 毫秒的最小值，但不同的单元具有不同的保持时间。在这种情况下，保持时间最短的单元被称为*弱单元*[45]。直观上，由于弱单元比其他单元泄漏性更强，似乎特别容易受到干扰误差的影响。相反，我们并未发现弱单元和受害单元之间存在任何强相关性。我们通过在向一个模块填充全 0 或全 1 之后，经过一段较长的时间（10 *秒*）既不访问也不刷新该模块来寻找其弱单元。如果在此过程中一个单元被损坏，我们就认为它是一个弱单元[45]。总的来说，我们能够为每个模块识别出 1M 个弱单元（984K、993K 和 1.22M），这与受害单元的数量相当。

17We comprehensively tested the first 32 rows in module A_{19} using hun-dreds of different random data patterns. Through statistical analysis on the experimental results, we were able to identify *almost certain* correlations be-tween a victim cell and the data stored in some other cells.

然而，仅有少数弱细胞也是受害细胞：700 个、220 个和 19 个。因此，我们得出结论，导致扰动误差的耦合路径可能与导致弱细胞的工艺变化无关。

受温度影响不大。当温度升高10℃时，每个单元的停留时间几乎减少了一半[39, 45]。为了看看这是否会大幅增加错误数量，我们在70.2.0℃下对三个模块进行了单次测试对，这比我们的默认环境温度高出20℃。与50℃下的迭代相比，错误数量没有显著变化：C10.2%，0.553%，C1.32%。我们还对三个模块在30.2.0℃下进行了单次测试对，结果相似：14.5%，C2.71%，5.11%。由此我们得出结论，干扰误差受温度影响不大。

8. 干扰误差的解决方案

我们研究了七种容忍、预防或减轻干扰误差的方法。每种方法在可行性、成本、性能、功耗和可靠性之间进行了不同的权衡。其中，我们认为我们的第七种也是最后一种解决方案，称为*PARA*，是最有效且开销最小的。第8.1节讨论了前六种解决方案。第8.2节详细分析了我们的第七种解决方案（PARA）。

8.1. 六个可能的解决方案

1. 制造更好的芯片。制造商可以通过改进电路设计来解决芯片层面的问题。然而，当工艺技术升级时，这个问题可能会再次出现。此外，随着电池变得更小且更脆弱，未来这个问题可能会变得更严重。

2. 纠正错误。服务器级系统采用带有额外 DRAM 芯片的 ECC 模块，会产生 12.5%的容量开销。然而，即使是这样的模块也无法纠正多比特干扰错误（第 6.3 节）。由于成本高昂，ECC 模块在消费级系统中很少使用。

3. 频繁刷新所有行。正如我们在第 6.2 节中所看到的，对于足够短的刷新间隔（$RI \leqslant RI_{th}$）），可以消除干扰误差。然而，频繁刷新也会降低性能和能效。当今的模块已经将其 1.4 - 4.5%的时间仅仅用于执行刷新[34]。如果刷新间隔缩短到 A_{20} 所要求的 8.2 毫秒，这一数字将增加到 11.0 - 35.0%。对于许多系统来说，如此之高的开销不太可能被接受。

4. 退役单元(制造商)。在 DRAM 芯片出售之前，制造商可以识别受害单元并将其重新映射到备用单元上[28]。然而，对所有受害单元进行详尽搜索可能需要几天甚至更长时间（第 7 节）。此外，如果受害单元数量众多，可能没有足够的备用单元供所有单元使用。

5. 退役单元(终端用户)。终端用户本身可以测试模块，并采用系统级技术来处理 DRAM 可靠性问题：禁用故障地址[2, 27, 62, 67]，将故障地址重新映射到预留地址[52, 53]，或者更频繁地刷新故障地址[44, 67]。然而，当模块中的每一行都是受害行时，第一/第二种方法是无效的（第 6.3 节）。另一方面，第三种方法效率低下，因为它总是更频繁地刷新受害行。

即使模块根本未被访问也是如此。在这三种方法中，最终用户都为识别和存储攻击者/受害者行的地址的成本买单。

*6. 识别"热点"行并刷新*相邻行。也许最直观的解决方案是识别频繁打开的行，并仅刷新其相邻行。挑战在于将硬件成本最小化以识别"热点"行。例如，当系统中有数百万行时，为每一行都设置一个计数器会过于昂贵。在其他领域，识别频繁项（从项流中）的广义问题已得到广泛研究。我们应用了一种著名的方法[37]，发现虽然它减少了计数器的数量，但它也要求对计数器进行昂贵的操作（例如，高度关联搜索）。我们还分析了一些近似方法，这些方法进一步减少了存储需求：布隆过滤器[11]、莫里斯计数器[50]及其变体[18, 21, 66]。然而，这些方法严重依赖于哈希函数，因此会引入哈希冲突。每当一个计数器超过阈值时，*许多*行就会被错误地标记为"热点"，导致对所有相邻行的刷新。

8.2. 第七种解决方案：PARA

我们防止 DRAM 干扰错误的主要提议是一种低开销机制，称为 *PARA*（概率*相邻行激活*）。PARA 的关键思想很简单：每次打开和关闭一行时，其相邻的一行也会以一定的低概率被打开（即刷新）。如果某一特定行碰巧被反复打开和关闭，那么从统计学上可以肯定的是，该行的相邻行最终也会被打开。PARA 的主要优点是*无状态*。PARA 不需要昂贵的硬件数据结构来计算行被打开的次数，也不需要存储攻击行/受害行的地址。

实现。PARA 在内存控制器中的实现方式如下。每当一行关闭时，控制器会抛一枚有偏差的硬币，出现正面的概率为 p，其中 $p \leqslant 1$。如果硬币出现正面，控制器会打开其相邻的一行，其中相邻两行中的任意一行被选择的概率相等（$p = 2$）。由于其概率特性，PARA *并*不能绝对保证相邻行总是能及时刷新。因此，PARA *无法*绝对确定地防止干扰错误。然而，其参数 p 可以设定得使干扰错误发生的概率极低——比其他系统组件的故障率低很多数量级（例如，每年有超过 1%的硬盘驱动器出现故障[54, 59]）。

错误率。我们通过考虑一种对抗性访问模式来分析 PARA 的错误概率，这种模式在刷新间隔期间打开和关闭一行恰好 (N_{th}) 次，但不多于 ☒ 次。每次关闭一行时，PARA 都会抛一枚硬币，并以概率 $p = 2$ 刷新给定的相邻一行。由于抛硬币是独立事件，对某一特定相邻行的刷新次数可以建模为一个随机变量 X，其具有参数为 $B(N_{th}$, 的二项分布。

$p = 2$)。只有在相邻行从未在任何一次N_{th}抛硬币（即 *XD0）期间被刷新时,才会发生错误*。这种事件发生的概率如下：. 1 p = 2/Nth 。当 *pD0.001* 时，我们在表 7 中针对不同的N_{th}值评估了该概率。该表显示了两种错误概率：一种是敌对访问模式持续 64 毫秒的概率，另一种是*一年的概率*。回想第 6.2 节，在我们的模块中，N_{th}的现实值为 139K - 284K 范围。对于 pD0.001 和N_{th}D100K，一年内发生错误的概率可以忽略不计，为 9:4 10^{14}。

持续时间	NthD50K	NthD100K	NthD200K
64毫秒	1:4 10^{11}	1:9 10^{22}	3:06 10 点 44 分
一年	6:8 10^{3}	9:4 10^{14}	1:8 10^{35}

表 7. 当 pD 为 *0.001* 时，PARA 的错误概率

邻接信息。对于PARA算法来说，内存控制器必须知道哪些行在物理上是相邻的。对于基于"热"行检测的替代解决方案（第8.1节）也是如此。没有这些信息，就无法选择性地刷新行，唯一的安全选择是盲目地刷新同一组中的*所有*行，这会导致性能损失。为了实现低开销的解决方案，我们主张制造商披露他们如何将逻辑行映射到物理行。这种*映射函数*可能很简单，比如指定逻辑行地址中用作物理行地址最低有效位的位偏移量。除了其他关于模块的元数据（例如容量和总线频率）外，映射函数可以存储在每个DRAM模块上存在的小型ROM（称为*SPD*）中[33]。制造商还应该披露他们如何重新映射有故障的物理行（第6.3节）。当有故障的物理行被重新映射时，映射到它的逻辑行会获得一组新的物理邻接行。SPD 还可以存储*重新映射函数,该函数*指定如何计算这些新物理邻域的逻辑行地址。考虑到重新映射的可能性，PARA 可以被配置为：*(i) 具有更高的 p 值;(ii)* 从一个更广泛的候选行池中选择要刷新的行，该候选行池包括重新映射的邻域以及原始邻域。

性能开销。我们使用一个周期精确的 DRAM 模拟器，评估了 PARA 对来自 SPEC CPU2006、TPC 和内存密集型微基准测试的 29 个单线程工作负载的性能影响（我们假设了一个合理的系统设置[41]，包括一个 4GHz 的无序核心和双通道 DDR3-1600）。由于重新映射，我们保守地假设一行最多可以有*十*个不同的行作为邻居，而不仅仅是两个。相应地，我们将 p 的值增加五倍到 0:005.20。在所有 29 个基准测试的平均中，在模拟的 100 毫秒期间，指令吞吐量仅下降了 0.197%。此外，在任何单个基准测试中，指令吞吐量的最大降幅为 0.745%。由此，我们得出结论

18Several patent applications propose to maintain an array of counters (“detection logic”) in either the memory controller [7, 8, 24] or in the DRAM chips themselves [6, 9, 23]. If the counters are tagged with the addresses of only the most recently activated rows, their number can be significantly reduced [24].

19Bains et al. [6] make the same argument. As an alternative, Bains et al. [7, 8] propose a new DRAM command called “targeted refresh”. When the memory controller sends this command along with the target row address, the DRAM chip is responsible for refreshing the row and its neighbors.

20We do not make any special considerations for victim cells with two aggressor rows (Section 6.3). Although they could be disturbed by either aggressor row, they could also be refreshed by either aggressor row.

我们认为，PARA 对性能的影响较小，这是合理的，原因在于其*(i)*强大的可靠性保证以及*(ii)*由于无状态特性导致的低设计复杂性。

9. 其他相关工作

干扰误差是一类常见的可靠性问题，不仅困扰着动态随机存取存储器（DRAM），也困扰着其他存储和存储技术：静态随机存取存储器（SRAM）[16, 26, 40]、闪存[10, 12, 13, 19, 25]和硬盘[36, 63, 68]。范德戈尔和德内夫[64]提出了一系列生产测试，可供 DRAM 制造商用于筛选有故障的芯片。其中一种测试是“锤子”测试，即每个单元被写入一千次，以验证它不会干扰附近的单元。2013 年，一家测试设备公司在 DDR4 DRAM（下一代商品 DRAM）的背景下提到了“行锤子”现象[48]。据我们所知，此前没有先前的研究展示和描述了来自现场的 DRAM 芯片中的干扰误差现象。

10 。结论

我们已经演示、表征并分析了现代商品 DRAM 芯片中的干扰误差现象。这些误差发生在对 DRAM 行的重复访问破坏了存储在其他行中的数据时。基于我们的实验表征，我们得出结论，干扰误差是一个新兴问题，可能会影响当前和未来的计算系统。我们提出了几种解决方案，包括一种新的无状态机制，通过概率性地刷新访问行旁边的行来提供对干扰误差的强统计保证。随着 DRAM 工艺技术缩小到更小的特征尺寸，我们希望我们的发现将能够促成新的系统级[51]方法来提高 DRAM 的可靠性。

致谢

我们感谢审稿人和 SAFARI 成员提供的反馈。我们感谢 IBM、英特尔和高通的支持。这项研究部分得到了 ISTC-CC、NSF（CCF 0953246、CCF 1212962 和 CNS 1065112）以及 SRC 的支持。Yoongu Kim 得到了英特尔奖学金的支持。

参考文献

[1] Memtest86+ v4.20. http://www.memtest.org.

[2] The GNU GRUB Manual. http://www.gnu.org/software/grub.

[3] Z. Al-Ars. *DRAM Fault Analaysis and Test Generation*. PhD thesis, TU Delft, 2005.

[4] Z. Al-Ars et al. DRAM-Specific Space of Memory Tests. In *ITC*, 2006.

[5] AMD. BKDG for AMD Family 15h Models 10h-1Fh Processors, 2013.

[6] K. Bains et al. Method, Apparatus and System for Providing a Memory Refresh. US Patent App. 13/625,741, Mar. 27 2014.

[7] K. Bains et al. Row Hammer Refresh Command. US Patent App. 13/539,415, Jan. 2 2014.

[8] K. Bains et al. Row Hammer Refresh Command. US Patent App. 14/068,677, Feb. 27 2014.

[9] K. Bains and J. Halbert. Distributed Row Hammer Tracking. US Patent App. 13/631,781, Apr. 3 2014.

[10] R. Bez et al. Introduction to Flash Memory. *Proc. of the IEEE*, 91(4), 2003.

[11] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7), 1970.

[12] Y. Cai et al. Error Patterns in MLC NAND Flash Memory: Measurement, Char-acterization, and Analysis. In *DATE*, 2012.

[13] Y. Cai et al. Program Interference in MLC NAND Flash Memory: Characteriza-tion, Modeling and Mitigation. In *ICCD*, 2013.

[14] S. Y. Cha. DRAM and Future Commodity Memories. In *VLSI Technology Short Course*, 2011.

[15] M.-T. Chao et al. Fault Models for Embedded-DRAM Macros. In *DAC*, 2009.

[16] Q. Chen et al. Modeling and Testing of SRAM for New Failure Mechanisms Due to Process Variations in Nanoscale CMOS. In *VLSI Test Symposium*, 2005.

[17] P.-F. Chia et al. New DRAM HCI Qualification Method Emphasizing on Repeated Memory Access. In *Integrated Reliability Workshop*, 2010.

[18] S. Cohen and Y. Matias. Spectral Bloom Filters. In *SIGMOD*, 2003.

[19] J. Cooke. The Inconvenient Truths of NAND Flash Memory. In *Flash Memory Summit*, 2007.

[20] DRAMeXchange. TrendForce: 3Q13 Global DRAM Revenue Rises by 9%, Sam-sung Shows Most Noticeable Growth, Nov. 12, 2013.

[21] L. Fan et al. Summary Cache: A Scalable Wide-Area Web Cache Sharing Proto-col. *Transactions on Networking*, 8(3), 2000.

[22] J. A. Fifield and H. L. Kalter. Crosstalk-Shielded-Bit-Line DRAM. US Patent 5,010,524, Apr. 23, 1991.

[23] Z. Greenfield et al. Method, Apparatus and System for Determining a Count of Accesses to a Row of Memory. US Patent App. 13/626,479, Mar. 27 2014.

[24] Z. Greenfield et al. Row Hammer Condition Monitoring. US Patent App. 13/539,417, Jan. 2, 2014.

[25] L. M. Grupp et al. Characterizing Flash Memory: Anomalies, Observations, and Applications. In *MICRO*, 2009.

[26] Z. Guo et al. Large-Scale SRAM Variability Characterization in 45 nm CMOS. *JSSC*, 44(11), 2009.

[27] D. Henderson and J. Mitchell. *IBM POWER7 System RAS*, Dec. 2012.

[28] M. Horiguchi and K. Itoh. *Nanoscale Memory Repair*. Springer, 2011.

[29] R.-F. Huang et al. Alternate Hammering Test for Application-Specific DRAMs and an Industrial Case Study. In *DAC*, 2012.

[30] Intel. Intel 64 and IA-32 Architectures Optimization Reference Manual, 2012.

[31] Intel. 4th Generation Intel Core Processor Family Desktop Datasheet, 2013.

[32] K. Itoh. Semiconductor Memory. US Patent 4,044,340, Apr. 23, 1977.

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

[34] JEDEC. *Standard No. 79-3F. DDR3 SDRAM Specification*, July 2012.

[35] M. K. Jeong et al. Balancing DRAM Locality and Parallelism in Shared Memory CMP Systems. In *HPCA*, 2012.

[36] W. Jiang et al. Cross-Track Noise Profile Measurement for Adjacent-Track Inter-ference Study and Write-Current Optimization in Perpendicular Recording. *Jour-nal of Applied Physics*, 93 10), 2003.

[37] R. M. Karp et al. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. *Transactions on Database Systems*, 28(1), 2003.

[38] B. Keeth et al. *DRAM Circuit Design. Fundamental and High-Speed Topics*. Wiley-IEEE Press, 2007.

[39] S. Khan et al. The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study. In *SIGMETRICS*, 2014.

[40] D. Kim et al. Variation-Aware Static and Dynamic Writability Analysis for Voltage-Scaled Bit-Interleaved 8-T SRAMs. In *ISLPED*, 2011.

[41] Y. Kim et al. A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM. In *ISCA*, 2012.

[42] Y. Konishi et al. Analysis of Coupling Noise between Adjacent Bit Lines in Megabit DRAMs. *JSSC*, 24(1), 1989.

[43] D. Lee et al. Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture. In *HPCA*, 2013.

[44] J. Liu et al. RAIDR: Retention-Aware Intelligent DRAM Refresh. In *ISCA*, 2012.

[45] J. Liu et al. An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms. In *ISCA*, 2013.

[46] L. Liu et al. A Software Memory Partition Approach for Eliminating Bank-level Interference in Multicore Systems. In *PACT*, 2012.

[47] J. A. Mandelman et al. Challenges and Future Directions for the Scaling of Dy-namic Random-Access Memory (DRAM). *IBM Journal of R&D*, 46(2.3), 2002.

[48] M. Micheletti. Tuning DDR4 for Power and Performance. In *MemCon*, 2013.

[49] D.-S. Min et al. Wordline Coupling Noise Reduction Techniques for Scaled DRAMs. In *Symposium on VLSI Circuits*, 1990.

[50] R. Morris. Counting Large Numbers of Events in Small Registers. *Communica-tions of the ACM*, 21(10), 1978.

[51] O. Mutlu. Memory Scaling: A Systems Architecture Perspective. In *MemCon*, 2013.

[52] P. J. Nair et al. ArchShield: Architectural Framework for Assisting DRAM Scal-ing by Tolerating High Error Rates. In *ISCA*, 2013.

[53] C. Nibby et al. Remap Method and Apparatus for a Memory System Which Uses Partially Good Memory Devices. US Patent 4,527,251, July 2 1985.

[54] E. Pinheiro et al. Failure Trends in a Large Disk Drive Population. In *FAST*, 2007.

[55] M. Redeker et al. An Investigation into Crosstalk Noise in DRAM Structures. In *MTDT*, 2002.

[56] K. Roy et al. Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits. *Proc. of the IEEE*, 91(2), 2003.

[57] K. Saino et al. Impact of Gate-Induced Drain Leakage Current on the Tail Distri-bution of DRAM Data Retention Time. In *IEDM*, 2000.

[58] J. H. Saltzer and M. F. Kaashoek. *Principles of Computer Design: An Introduc-tion*. Chapter 8, p. 58. Morgan Kaufmann, 2009.

[59] B. Schroeder and G. A. Gibson. Disk Failures in the Real World: What Does an MTTF of 1,000, 000 Hours Mean to You? In *FAST*, 2007.

[60] N. Suzuki et al. Coordinated Bank and Cache Coloring for Temporal Protection of Memory Accesses. In *ICESS*, 2013.

[61] A. Tanabe et al. A 30-ns 64-Mb DRAM with Built-In Self-Test and Self-Repair Function. *JSSC*, 27(11), 1992.

[62] D. Tang et al. Assessment of the Effect of Memory Page Retirement on System RAS Against Hardware Faults. In *DSN*, 2006.

[63] Y. Tang et al. Understanding Adjacent Track Erasure in Discrete Track Media. *Transactions on Magnetics*, 44(12), 2008.

[64] A. J. van de Goor and J. de Neef. Industrial Evaluation of DRAM Tests. In *DATE*, 1999.

[65] A. J. van de Goor and I. Schanstra. Address and Data Scrambling: Causes and Impact on Memory Tests. In *DELTA*, 2002.

[66] B. Van Durme and A. Lall. Probabilistic Counting with Randomized Storage. In *IJCAI*, 2009.

[67] R. Venkatesan et al. Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM. In *HPCA*, 2006.

[68] R. Wood et al. The Feasibility of Magnetic Recording at 10 Terabits Per Square Inch on Conventional Media. *Transactions on Magnetics*, 45(2), 2009.

[69] Xilinx. *Virtex-6 FPGA Integrated Block for PCI Express*, Mar. 2011.

[70] Xilinx. *ML605 Hardware User Guide*, Oct. 2012.

[71] Xilinx. *Virtex-6 FPGA Memory Interface Solutions*, Mar. 2013.

[72] J. H. Yoon et al. Flash & DRAM Si Scaling Challenges, Emerging Non-Volatile Memory Technology Enablement. In *Flash Memory Summit*, 2013.

[73] T. Yoshihara et al. A Twisted Bit Line Technique for Multi-Mb DRAMs. In *ISSCC*, 1988.