可抵御缓存侧信道攻击的随机化缓存设计

宋威，薛子涵

中国科学院信息工程研究所
信息安全国家重点实验室

RISC-V技术及生态研讨会　2023年8月22日

SKLOIS 信息安全国家重点实验室

中国科学院信息工程研究所 INSTITUTE OF INFORMATION ENGINEERING,CAS

主要内容

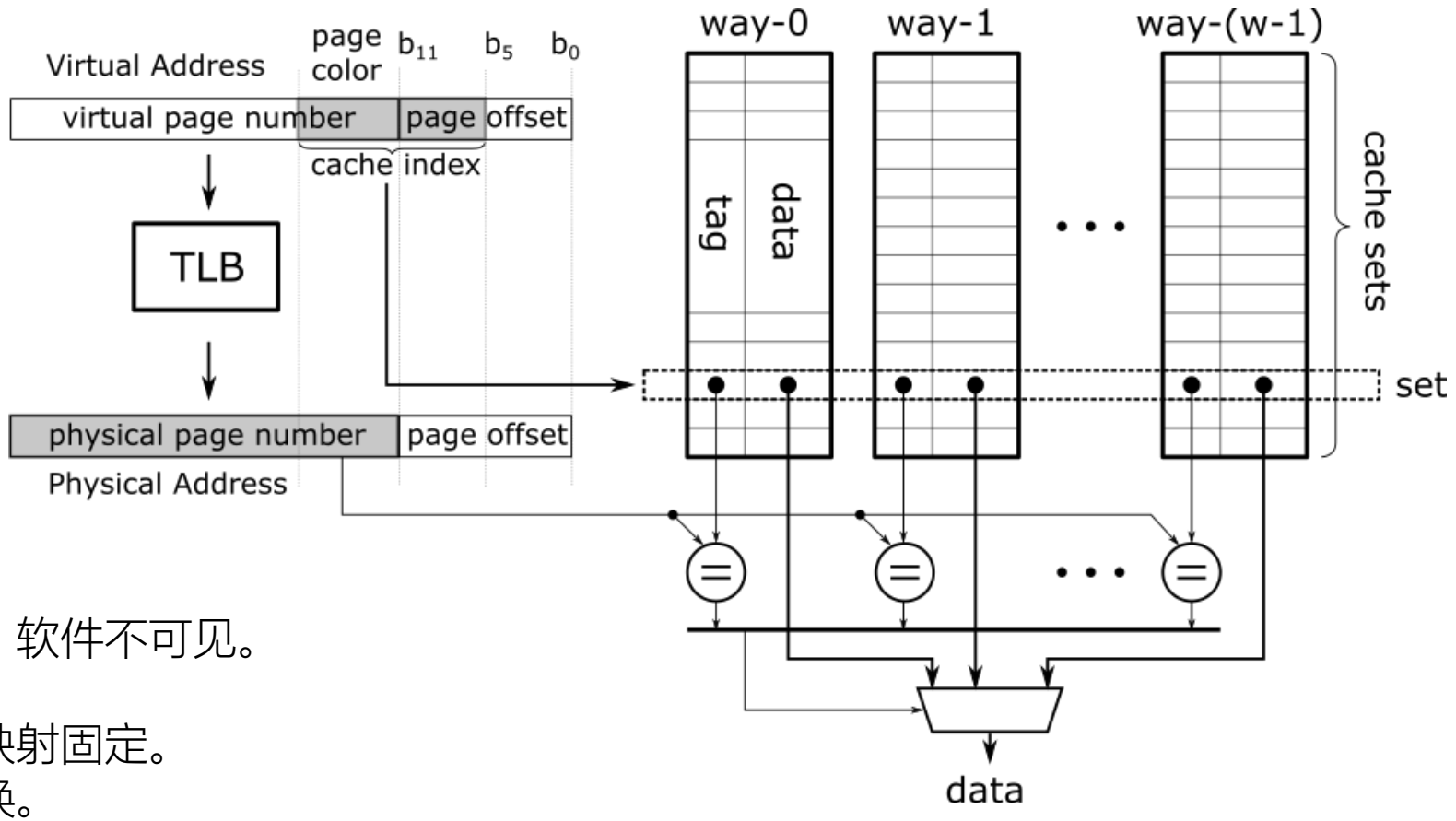
缓存侧信道攻击是攻击者通过在受害机器上运行恶意程序（非接触式）从缓存中获取关键信息的一种信息泄露的攻击方式。近年来被大量采用。

缓存随机化是近年来提出的一种纯硬件的缓存防御方法，可以抵御冲突型的缓存侧信道攻击。

现在还没有任何一个实现缓存随机化防御的真实处理器。那么，我们来做一个！

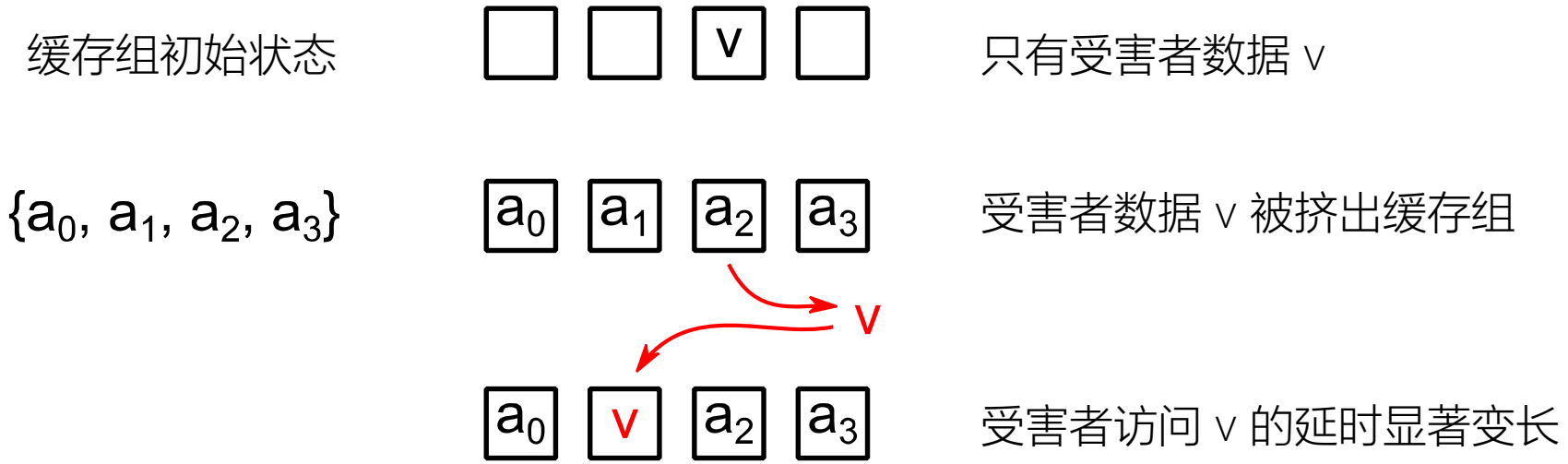
- 冲突型缓存侧信道攻击
- 缓存随机化防御
- 我们的优化
- 我们在Rocket-Chip上的实现

缓存的基本结构



- 缓存是微体系结构模块，软件不可见。
- 缓存被多核共享。
- 缓存内地址到缓存组的映射固定。
- 当缓存组满，会发生置换。
- 置换算法已知：LRU，Random。

冲突型缓存侧信道攻击：Prime+Probe



- 正常情况下，受害者的目标数据 v 在缓存命中
- 攻击者通过访问一个驱逐集 $\{a_0, a_1, a_2, a_3\}$，强迫将 v 挤出缓存.
- 受害者再次访问 v 的延时显著变长。

$\{a_0, a_1, a_2, a_3\}$ 和 v 在缓存中保存在同一个缓存组 (congruent)

（最小）驱逐集：一个包含足够多congruent地址的集合，访问它可以将目标地址驱逐出缓存。

驱逐集寻找算法

在早期的电脑上，驱逐集是算出来的。在现在的电脑上，驱逐集是找出来的。（原因：虚拟地址到物理地址映射被保护，Intel的复杂末级缓存分片寻址方式）

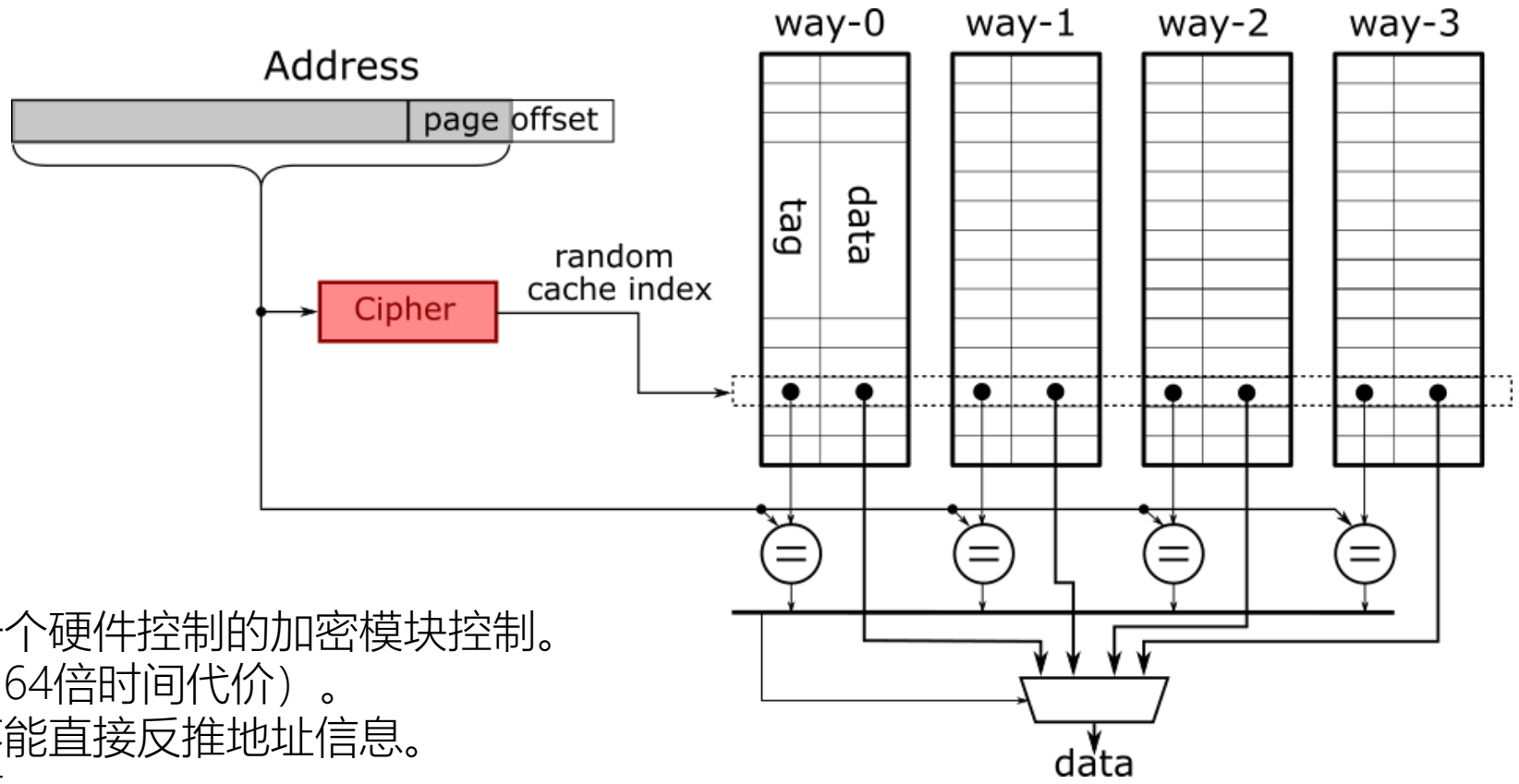
- 组消除[1,2,3]（GE: Group Elimination）：80ms, 80%
- 覆盖-裁剪-触发算法[3,4]（PPP: Prime, Prune and Probe）：1.4ms, 24%

- 冲突测试[3]（CT: Conflict Testing）：18ms, 70%

其他：W+W[5]（10ms, 5%），CTPP[6]（1.3ms, 92%）

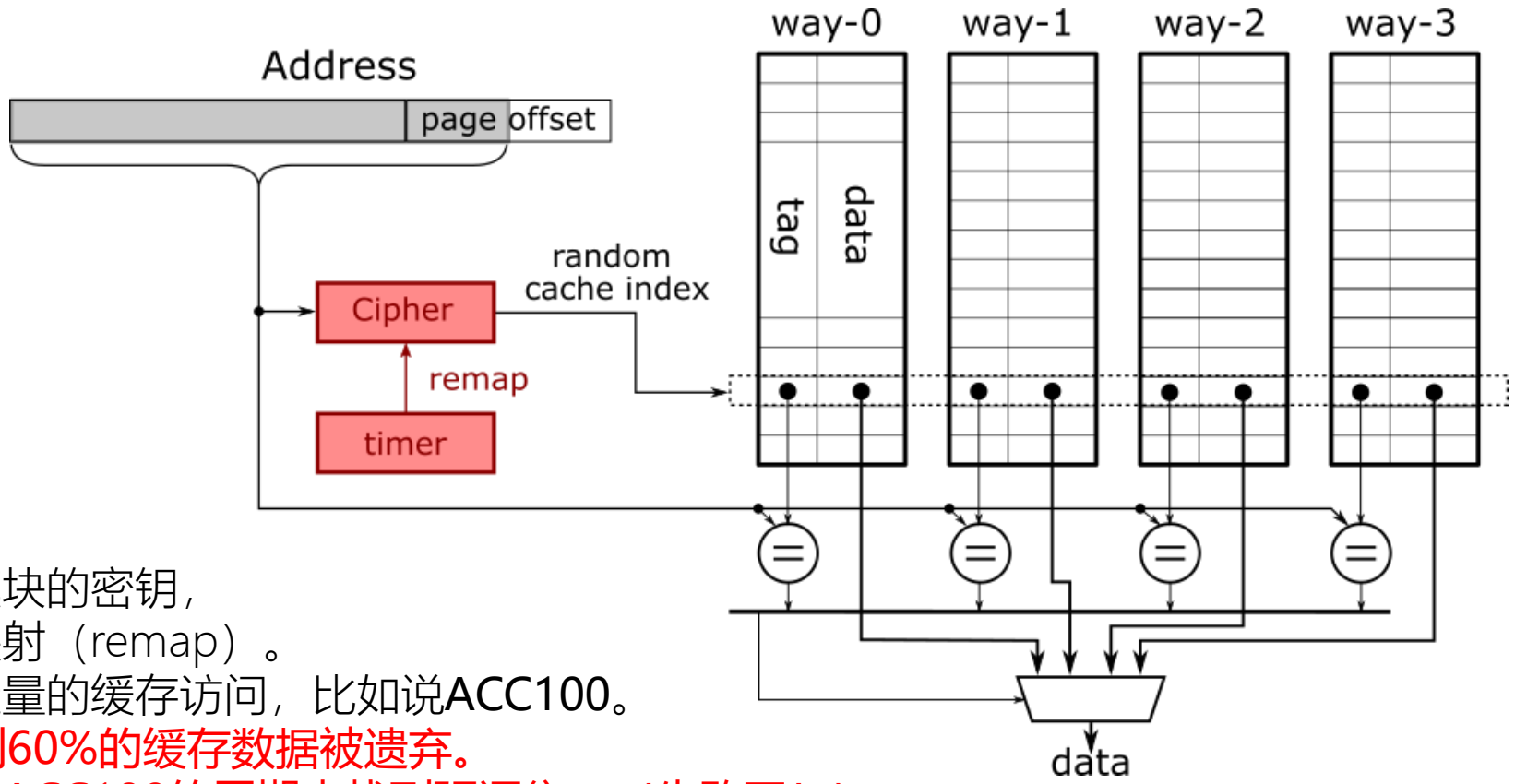
[1] P. Vila, B. Köpf, J. F. Morales. "Theory and practice of finding eviction sets." S&P'19.
[2] W. Song, P. Liu. "Dynamically finding minimal eviction sets can be quicker than you think for side-channel attacks against the LLC." RAID'19.
[3] M. K. Qureshi. "New attacks and defense for encrypted-address cache." ISCA'19.
[4] A. Purnal, I. Verbauwhede. "Advanced profiling for probabilistic Prime+Probe attacks and covert channels in ScatterCache." ArXiv, Aug 2019.
[5] J. P. Thoma, T. Güneysu. "Write me and I'll tell you secrets – Write-after-write effects on Intel CPUs." RAID'22.
[6] Z. Xue, J. Han, W. Song. "CTPP: A fast and stealth algorithm for searching eviction sets on Intel processors." RAID'23.

静态缓存随机化



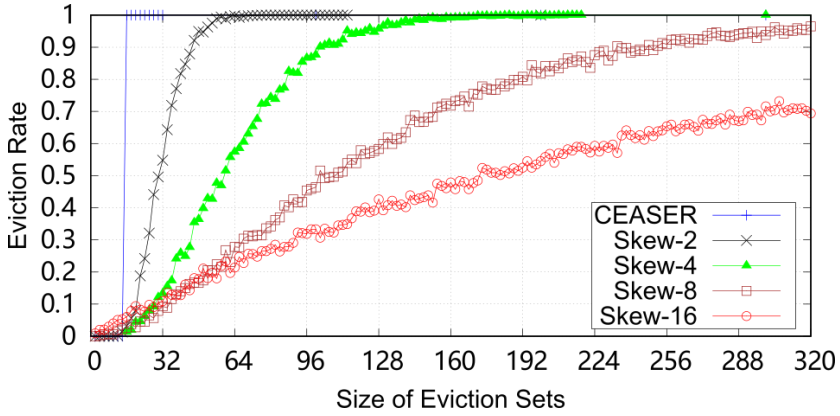
- 地址到缓存组的映射由一个硬件控制的加密模块控制。
- 页内偏移也不能被利用（64倍时间代价）。
- 即使找到了驱逐集，也不能直接反推地址信息。
- 寻找攻击目标靠缓存扫描。
- 攻击者仍然可以构造驱逐集！

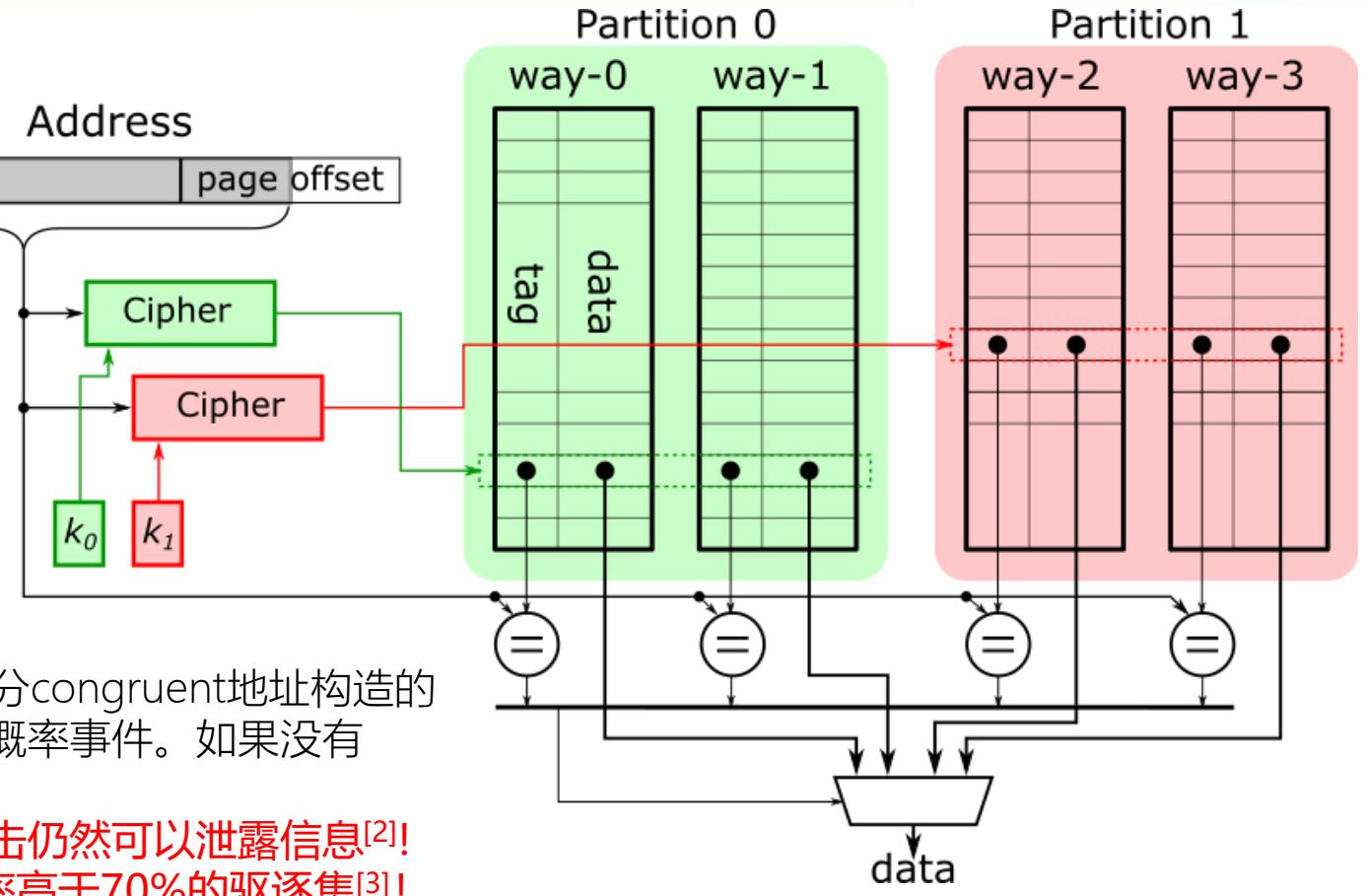
动态缓存随机化



- 周期性的修改硬件加密模块的密钥，
  更新缓存映射，也叫重映射（remap）。
- 重映射的周期一般为一定量的缓存访问，比如说ACC100。
- 每次remap会造成40%到60%的缓存数据被遗弃。
- 安全假设：攻击者不能在ACC100的周期内找到驱逐集。（失败了！）
  CT能在ACC30左右找到驱逐集，即使将remap周期降到ACC20，CT仍然能在几毫秒内找到驱逐集！

动态随机化Skewed缓存

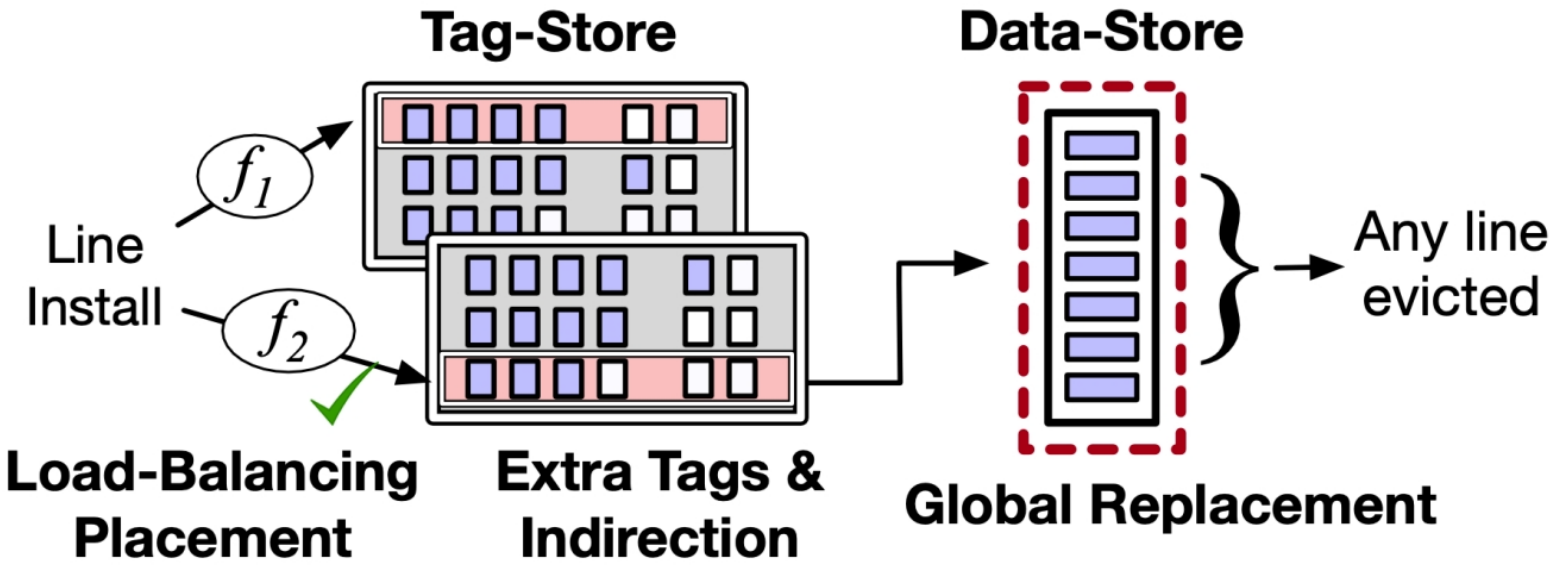




- 引入Skewed缓存[1]：攻击者只能构造由部分congruent地址构造的驱逐集。使用驱逐集挤出目标地址变成了概率事件。如果没有clflush指令，驱逐集只能使用一次。
- 即使使用成功极低的驱逐集，利用重复攻击仍然可以泄露信息[2]!
- 在ACC100内，攻击者仍然可以构造成功率高于70%的驱逐集[3]!

[1] M. K. Qureshi. "New attacks and defense for encrypted-address cache." ISCA'19.
[2] T. Bourgeat, J. Drean, Y. Yang, et al. "CaSA: End-to-end quantitative security analysis of randomly mapped caches." MICRO'20.
[3] W. Song, B. Li, Z. Xue, et al. "Randomized last level caches are still vulnerable to cache side channel attacks! But we can fix it." S&P'21.

MIRAGE



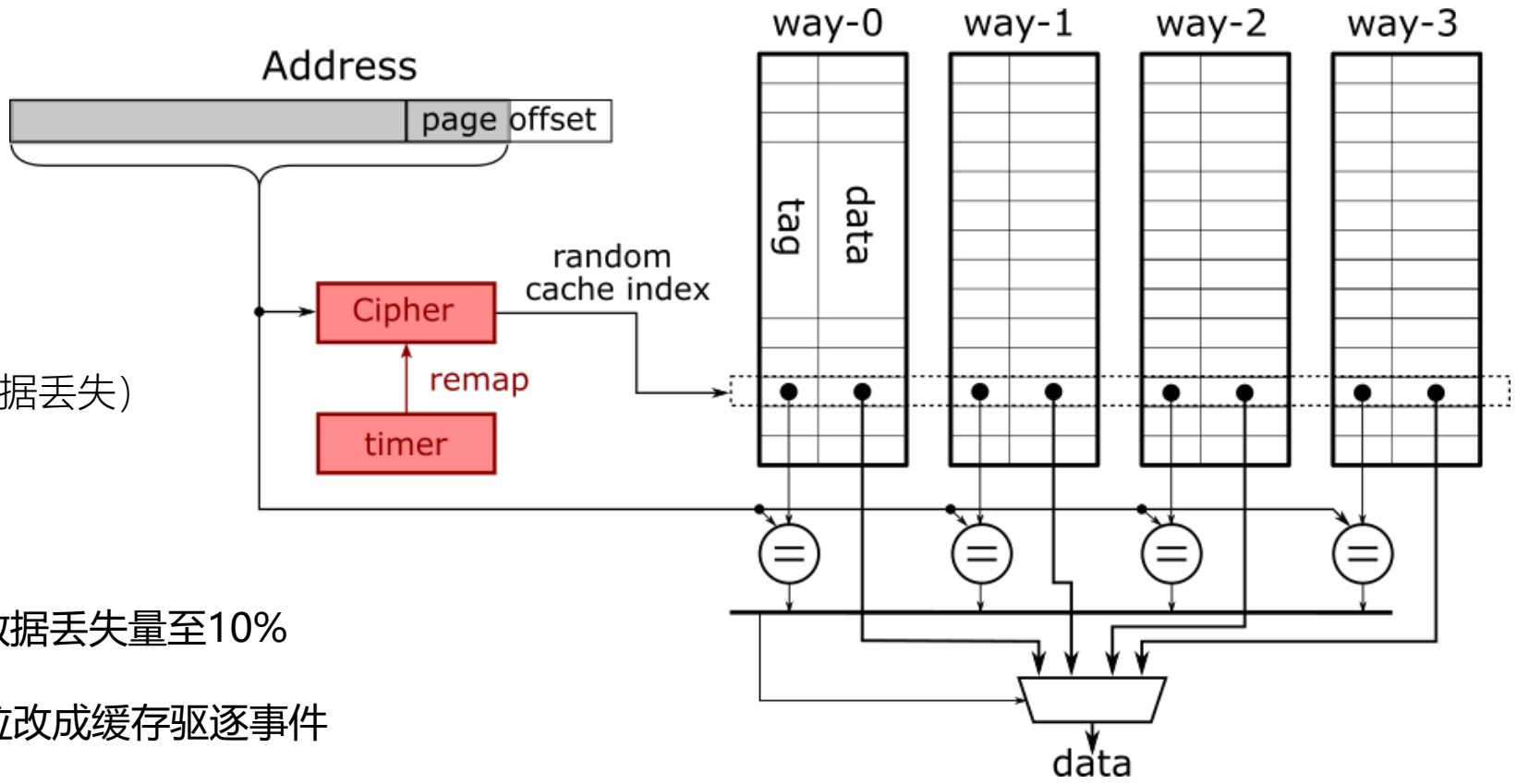
G. Saileshwar, M. Qureshi. "MIRAGE: Mitigating conflict-based cache attacks with a practical fully-associative design." Security'21.

- 目标：彻底消除攻击者控制的associative eviction。
- 方法：Skewed缓存+元数据冗余供给+skew平衡分布+元数据和数据存储分离。
  - Skewed缓存：缓存随机化和多种缓存映射，提供原始随机分布来源。
  - 元数据冗余供给+skew平衡分布：极度降低由元数据组满导致的associative eviction。
  - 元数据和数据存储分离：降低数据存储空间，并利用数据存储达到全局随机置换。
- 攻击者构造一次associative eviction需要上万年？！
- 22%的SRAM存储空间代价！（运行时性能代价3%？）

我们真的需要这样的缓存架构吗？

我们的构想：**动态随机化传统多路组相联缓存**

影响性能的安全方案往往不会被接受，只有性能影响小，可在现有架构上打补丁的安全方案才容易被现有处理器采用！

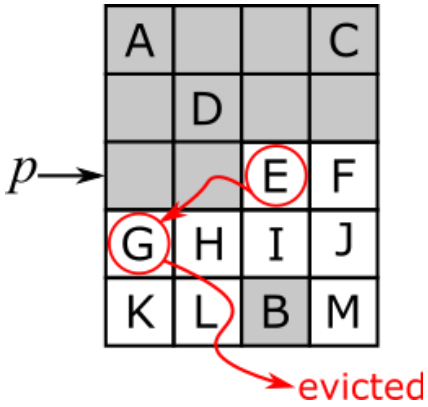


- 问题：
  Remap的代价高（40~60%数据丢失）
  Remap过于频繁（ACC10）

- 我们的解决方案：
  - Remap的代价
    引入多步置换链，降低数据丢失量至10%
  - Remap的频率
    将remap周期的计数单位改成缓存驱逐事件
    引入攻击检测器

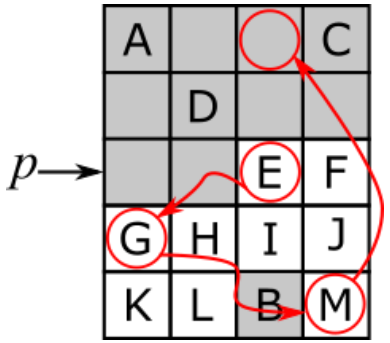
- 性能代价：10.3% logic area，0.5% SRAM，< 1.0% CPI, 2% Power

利用多步置换链降低数据丢失



传统remap：
按顺序对所有缓存块进行移位。
当新的缓存组没有空闲位置时，按照置换算法选择一个缓存块（G），将其驱逐。

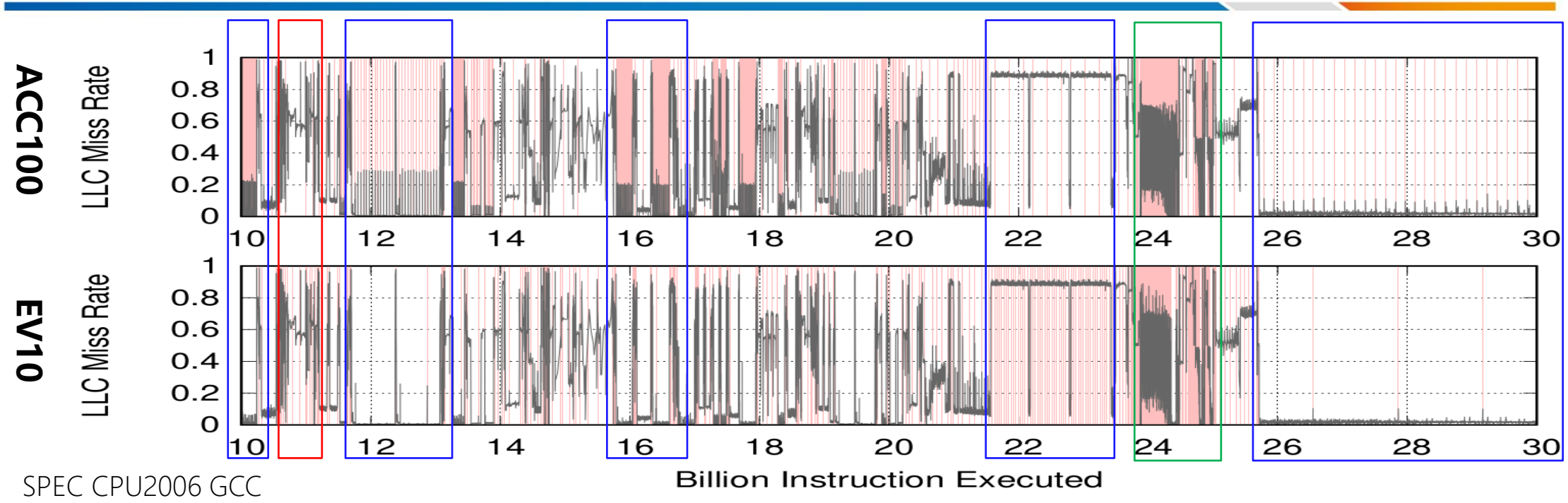
结果：50~60%的缓存块在remap过程中丢失。



多步置换链：
当新的缓存组没有空闲位置时，按照置换算法选择一个缓存块（G），将缓存块（E）存入。
继续将被置换出的缓存块（G）作为移位目标，继续移位，直至找到一个空闲位置，或者目标缓存组的所有缓存块都已经被remap。

结果：~10%的缓存块在remap过程中丢失。

替换remap周期计数单位为缓存驱逐事件

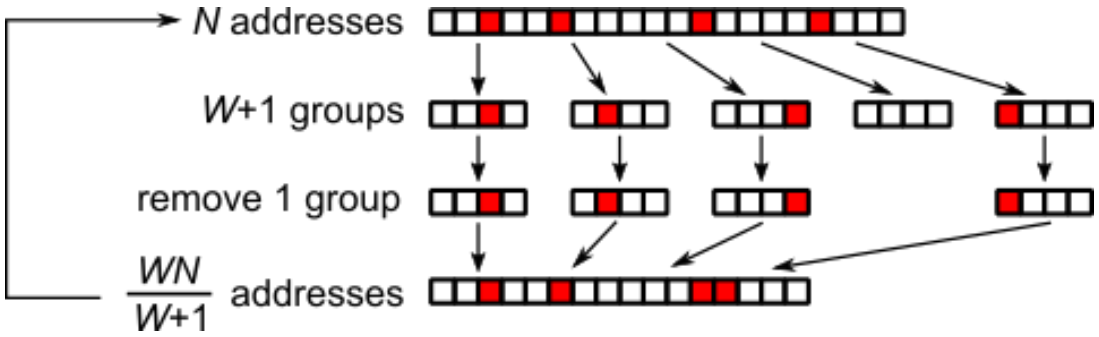


SPEC CPU2006 GCC

- 由于缓存的过滤特性，当缓存足够预热，在攻击时末级缓存的**访问数**和**驱逐数**是**相等**的！
- 对于正常程序来说，缓存性能好意味着命中率高，**缓存驱逐率极低**！
- 当缓存驱逐率高时，ACC和EV计数都会造成高的remap频率，但是**缓存本身的性能已经很低**！
- 如果缓存的随机映射导致了更多的冲突，这些冲突引发remap可能会导致一个更好的映射，提高缓存性能！

攻击检测器：GE算法

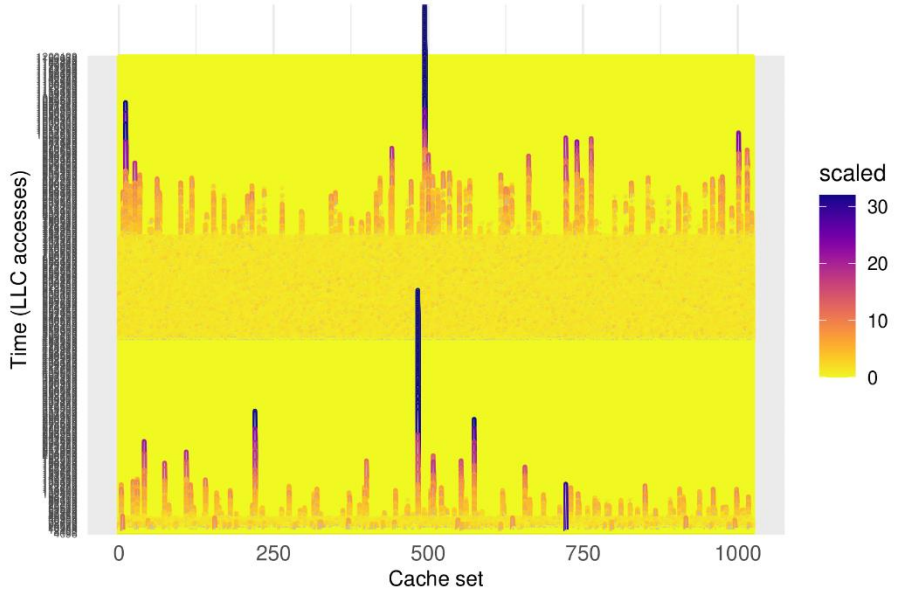
- GE算法：
  1. 找一个足够大（N）的候选地址集。
  2. 将候选集随机分成W+1组。由于驱逐集只需要W个元素，一定有一组地址是多余的。
  3. 通过测试这W+1各组，找出多余的那一组，将其删除。
  4. 如果留下的地址数量大于W，回到第2步。



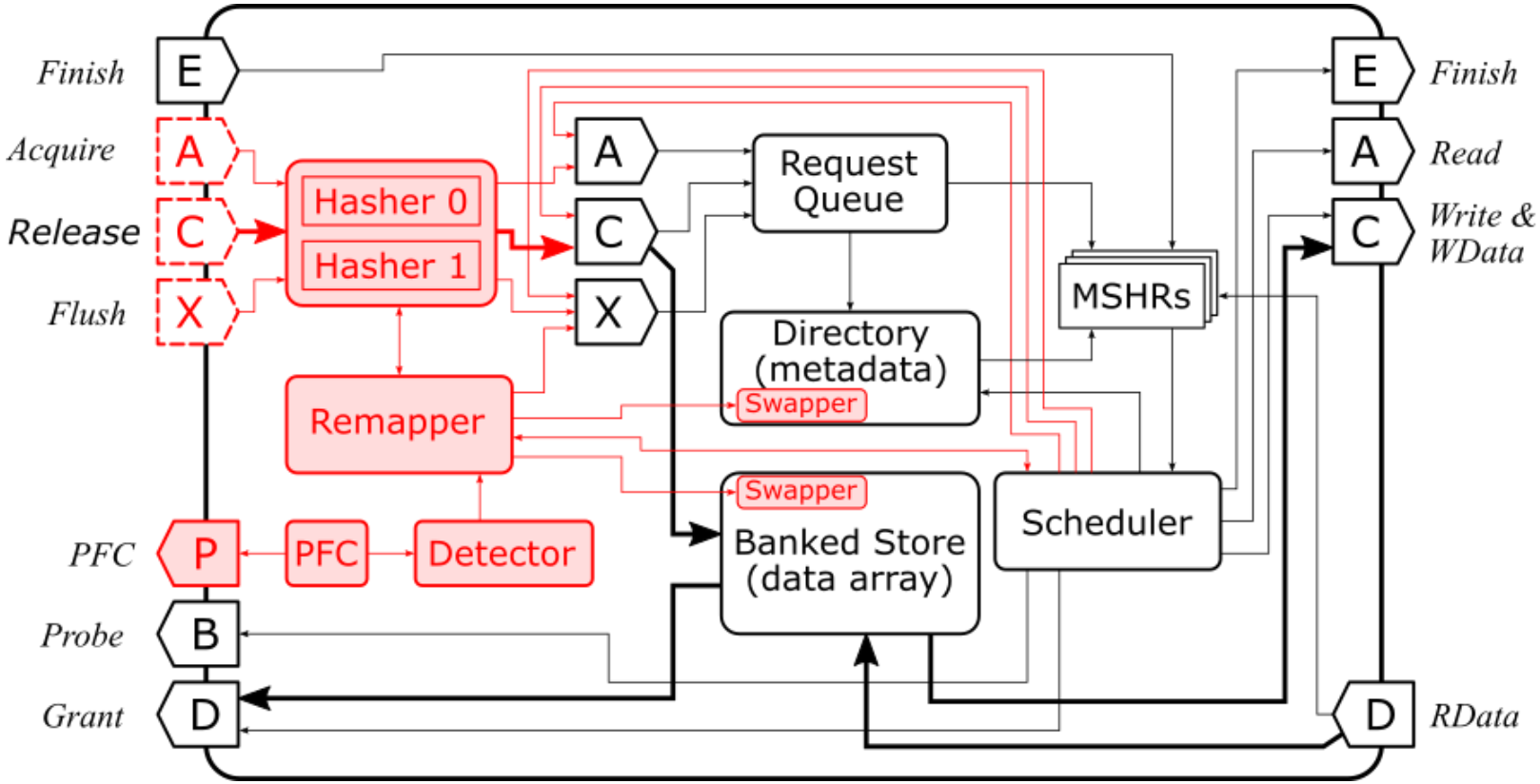
考虑这么一个问题：对于这样的递归的算法，在缓存上的访问行为是否有特征？

使用Z-Score对缓存组上的置换事件做分布标准化。

所有的驱逐集寻找算法都不可避免地在目标缓存组上造成异常高频的缓存冲突，该特征不可避免！



在Rocket-Chip上的实现



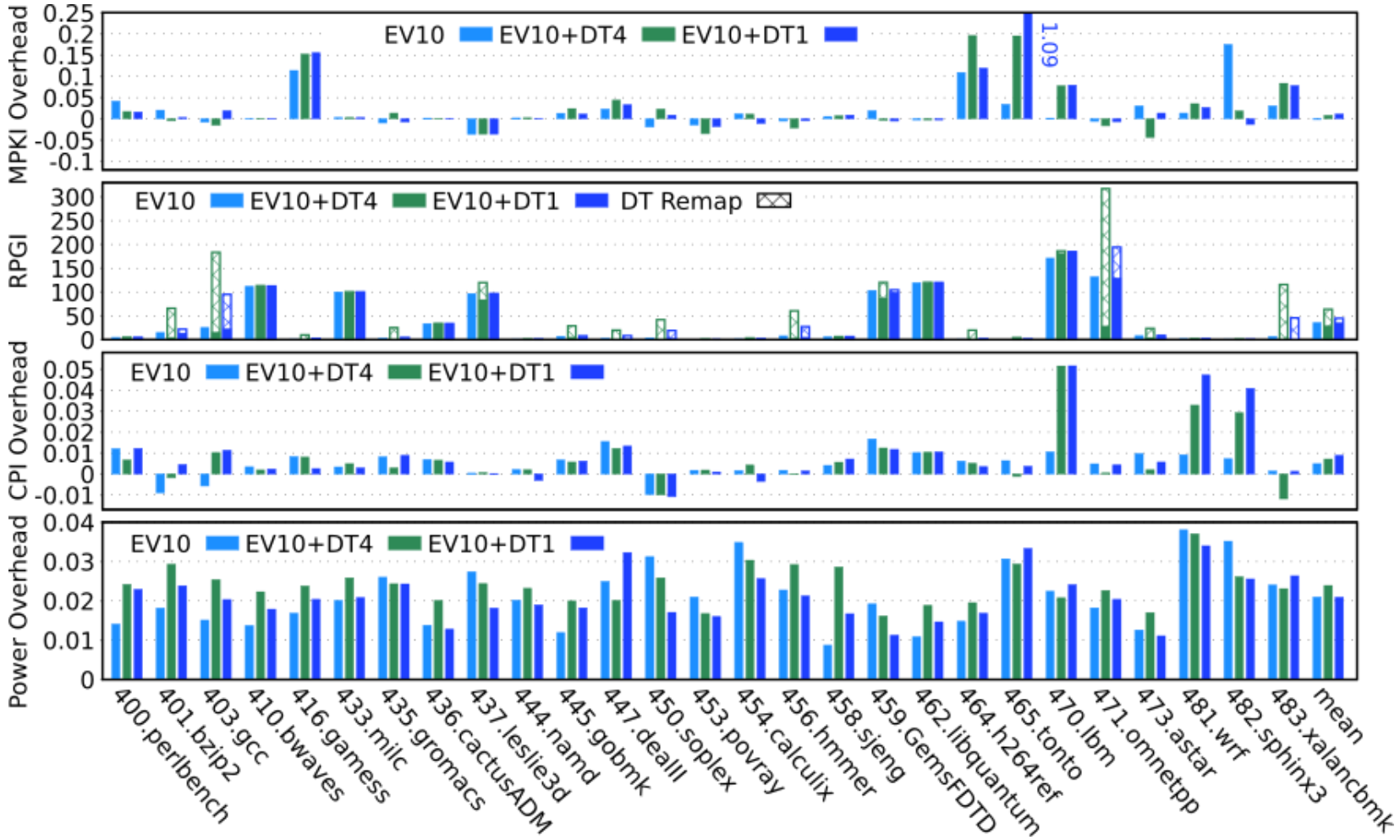
防御效果

	Detector Combination					
Alg.	**Static**	**EV10**	**DT4**	**DT1**	**EV10+DT4**	**EV10+DT1**
GE	97.2%	~0.0%	~0.0%	~0.0%	~0.0%	~0.0%
PPP	20.4%	13.3%	~0.0%	~0.0%	~0.0%	~0.0%
CT	14.8%	~0.0%	~0.0%	~0.0%	~0.0%	~0.0%
CT-fast	13.9%	2.5%	1.0%	0.1%	0.3%	~0.0%
W+W	17.4%	11.7%	~0.0%	~0.0%	~0.0%	~0.0%

面积代价 （FPGA）

		Slice	Percent	Overhead	RAM	Percent
Original	Rocket-Chip	24287			354.5	
	LLC	3114	12.8%		272.5	76.7%
	Channels	802	3.30%		0	
	MSHRs	823	3.39%		0	
	Request Queue	183	0.75%		0	
	Banked Store	1099	4.53%		256	71.9%
	Directory	392	1.61%		16.5	4.63%
Random	Rocket-Chip	28115		15.8%	356	
	LLC	5609	20.0%	80.1%	274	77.0%
	Channels	1076	3.83%	34.2%	0	
	MSHRs	910	3.24%	10.6%	0	
	Request Queue	311	1.11%	69.9%	0	
	Banked Store	1641	5.84%	49.3%	256	71.9%
	Data Swapper	416	1.48%		0	
	Directory	540	1.92%	37.8%	16.5	4.63%
	Meta Swapper	190	0.68%		0	
	Hasher	526	1.87%		0	
	Detector	362	1.29%		1.5	0.42%
	Remapper	66	0.23%		0	
	PFC	485	1.73%		0	

性能代价


MPKI Overhead
EV10
EV10+DT4
EV10+DT1
1.09
RPGI
DT Remap
CPI Overhead
Power Overhead
400.perlbench
401.bzip2
403.gcc
410.bwaves
416.gamess
433.milc
435.gromacs
436.cactusADM
437.leslie3d
444.namd
445.gobmk
447.dealII
450.soplex
453.povray
454.calculix
456.hmmer
458.sjeng
459.GemsFDTD
462.libquantum
464.h264ref
465.tonto
470.lbm
471.omnetpp
473.astar
481.wrf
482.sphinx3
483.xalancbmk
mean

总结

- 缓存随机化可以用来抵御冲突型的缓存侧信道攻击
- 观点：
  - 基于skewed缓存的随机化方案性能代价过高。
  - 完全消除associative eviction是没有必要的。
  - 传统的多路组相联缓存可以做到足够安全。
- 我们的改进
  - 基于置换链的重随机
  - 使用缓存驱逐事件为重随机周期计量单位
  - 使用基于Z-Score的检测算法在线检测攻击
- 我们在Rocket-Chip上实现了末级缓存随机化
  - 10.3% logic area，0.5% SRAM，< 1.0% CPI, 2% Power

谢谢！

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS