Optimizing Cauchy Reed-Solomon Coding via ReRAM Crossbars in SSD-based RAID Systems

Lei Han, Shangzhen Tan, Bin Xiao, Chenlin Ma, Zili Shao

摘要

诸如 Cauchy Reed-Solomon 码这样的擦除码在基于 SSD 的 RAID 阵列中的容错方面的重要性越来越大。然而，基于处理器的 RAID 控制器上的擦除编码依赖于 Galois Filed 算术来执行矩阵-向量乘法，这增加了计算的复杂性并导致大量的内存访问。在本文中，我们研究了利用 ReRAM 来提高擦除编码的性能。我们提出了 Re-RAID，它在 RAID 和 SSD 控制器中使用 ReRAM 作为主存储器，其中擦除编码可以在 ReRAM 上处理。我们还提出了一个混合的 Cauchy-Vandermonde 矩阵作为编码的生成矩阵。通过这样做，ReRAID 可以将单个故障的重建任务分配给 SSD，然后 SSD 可以用 ReRAM 内存恢复数据。

关键词：ReRAM；SSD；RAID；Cauchy Reed-Solomon

1 引言

基于 SSD 的 RAID 阵列已经变得无处不在。基于 SSD 的 RAID 阵列所采用的奇偶校验码能够保护数据并保持高水平的服务性能。在 RAID 系统中生成奇偶校验码的方法是多种多样的，如镜像和擦除编码。越来越多的 RAID 系统已经采用了擦除编码来保护数据，因为与其他编码策略相比，擦除编码的存储开销较低。里德-所罗门（RS）编码是擦除编码中最流行的，它通过矩阵乘法进行编码和解码[1]。RS 编码在云存储系统中得到了广泛的应用。

然而，基于矩阵乘法的 Reed-Solomon 编码对目前基于处理器的实现方式来说非常有挑战性。首先，矩阵乘法的实现涉及到伽罗瓦场（GF）算法[2]，对于 w 位的字则称为 GF(2^w)。此算法在处理器上计算需要非常昂贵的代价。GF(2^w) 的乘法和除法运算依赖于乘法表或离散对数。Cauchy Reed-Solomon（CRS）码[3]将 RS 码覆盖为 1 位字的码，它用额外的异或操作取代了昂贵的乘法。然而，它仍然是挑剔的，因为处理器需要特殊的 MMX 和 SSE 指令集扩展来提高速度。其次，目前基于处理器的实现方式将数据编码分离为内存访问和处理器处理，这导致了大量的数据在内存和处理器之间移动。在基于 GPU 的解决方案[3]中，它通过从恒定内存中读取比特矩阵和从共享内存中读取数据来加速 CRS 编码。第三，由于修复问题，RS 编码的重建在 RAID 系统中是次优的[4]。即使条带中的一个数据块出现故障，其他具有相同数量数据块的幸存者也需要被转移到 RAID 控制器进行重建，在修复带宽和磁盘 I/O 方面产生许多倍的开销。

一种新兴的非易失性存储器，金属氧化物电阻式随机存取存储器（ReRAM），具有在数据存储中进行算术运算的能力。ReRAM 具有低延迟、低能耗和出色的耐久性。此外，ReRAM 可以在交叉开关结构中高效地执行矩阵-矢量乘法和求和运算，这本质上适合 CRS 编码。ReRAM 交叉结构已被广泛研究，用于加速几种应用，包括图处理[5-6]和神经网络训练[7-8]。

本文提出了一个新颖的 ReRAM 优化的 RAID 系统来加速 CRS 编码，称为 ReRAID。首先，ReRAID 使用 ReRAM 作为 RAID 和 SSD 控制器的替代主存储器。ReRAM 具有低延迟和低能耗的特点，是主存储器的良好候选者。除了作为存储存储器外，其中一部分 ReRAM 存储器被用来加速擦除编码。其次，为了减轻 RAID 控制器的计算工作量，论文提出了一个混合的 Cauchy-Vandermonde 矩阵作为编码的生成矩阵，通过这个矩阵，第一个奇偶校验码是与所有数据块的位数异或相加的结果。当单个故障发生时，ReRAID 可以将重建任务分配给其他幸存的数据盘，这些数据盘可以利用 ReRAM 内存，通过与第一奇偶校验码和其他幸存的数据块在条带内进行异或求和来恢复丢失的数据。

2 Motivation

原论文研究的课题为利用 ReRAM 加速擦除码 (Erasure Code) 的计算，与本人的未来研究方向强相关。通过复现原论文的工作，可以帮助学习 ReRAM。如图 1所示，在存储层次结构中，内存为了适配 CPU 和缓存，要求有极低延迟；外存作为存放数据的主要介质，要求能够做到大容量和持久化。而 ReRAM 作为一种新兴的非易失性存储器，能够同时达到内存和外存两者的要求。并且，ReRAM 的 Crossbar 结构能够进行向量-矩阵运算。因此，用 ReRAM 作为主存，可以减少在运算过程中的数据迁移，同时加速运算，这是本人未来研究方向的一个重点。

但是受制于硬件条件，现有的基于的 ReRAM 对不同应用进行加速的工作都是通过模拟器实现的。与对神经网络或图处理进行加速的工作相比，利用 ReRAM 加速 Erasure Code 的数据映射会相对简单，有利于了解 ReRAM 的开发方式，为之后的科研工作打好基础。

此外，擦除码通过对输入数据进行编码生成校验码，这虽然会带来一定程度的数据冗余，但是却提高了数据的可靠性。当文件故障或其他意外导致数据丢失时，利用校验码可以解码得到原始数据，降低数据损坏或丢失的风险。这种在数据冗余和可靠性之间的取舍 (Trade-off) 值得学习。

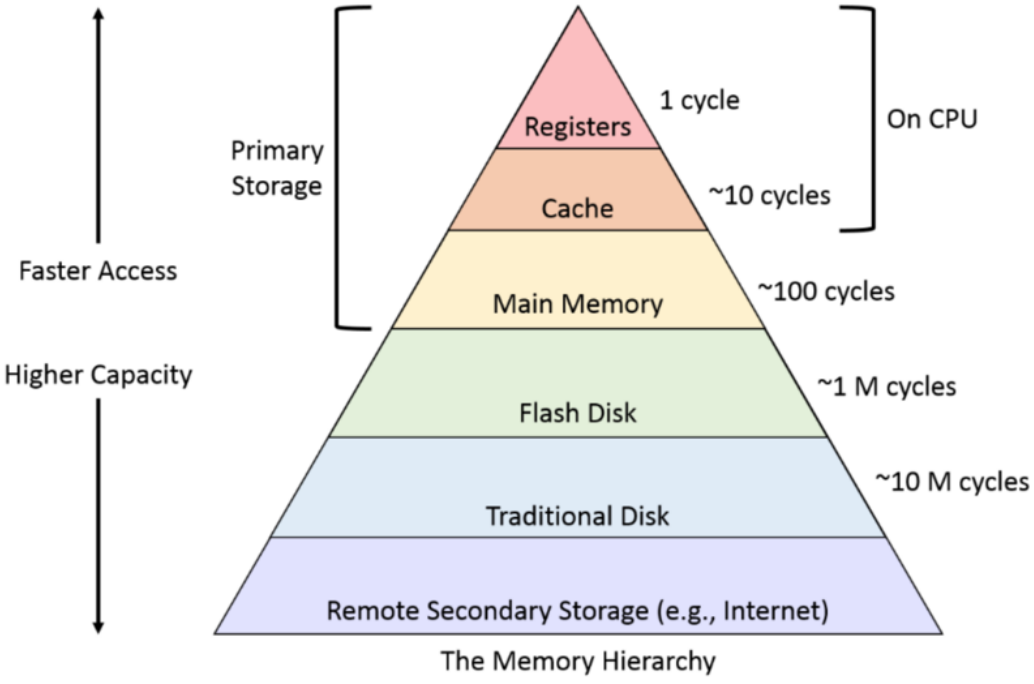


图 1: 存储层次结构

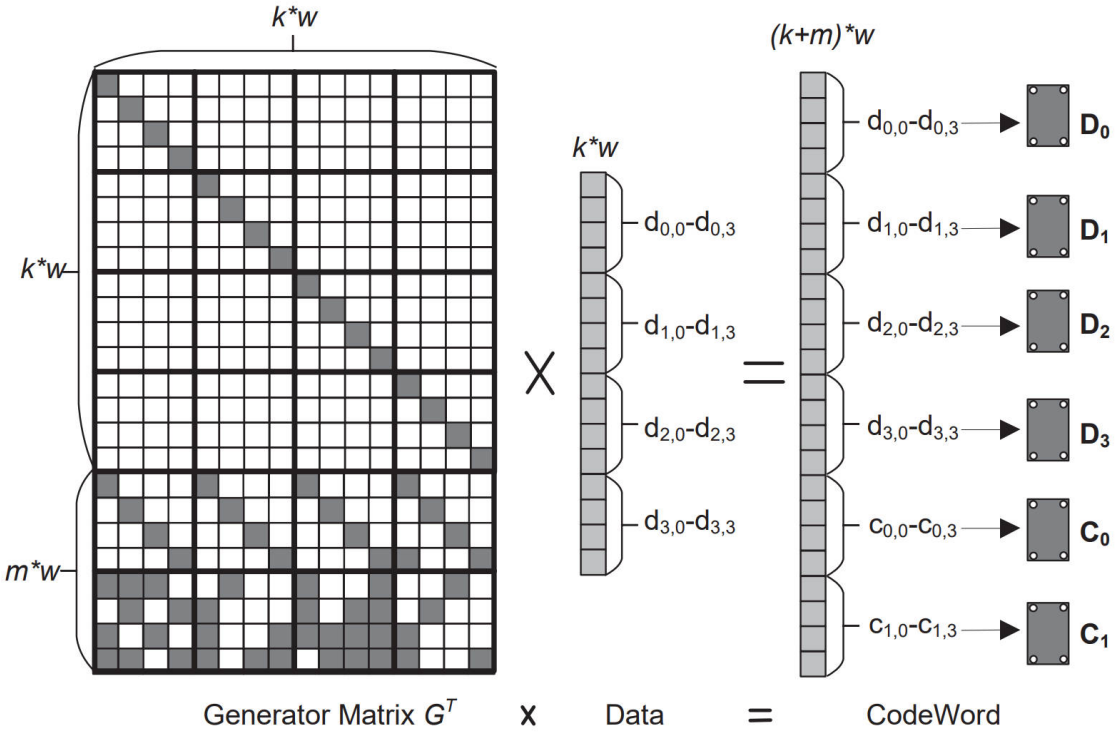


图 2: CRS 码的矩阵-向量表示，其中 $k=4$，$m=2$，$w=4$。每个元素是 1bit

3 相关工作

3.1 基于 SSD 的 RAID 系统

基于固态硬盘的 RAID 存储系统由一个 RAID 控制器和多个固态硬盘组成。RAID 系统增加了容错冗余，奇偶校验数据由 RAID 控制器生成。在存储阵列中，单次故障的频率远远高于多次故障，占故障的 90% 以上[9]。典型的固态硬盘包括 SSD 控制器、DRAM 缓冲区和连接到闪存的闪存控制器。基于 DRAM 的缓冲区用于缓存读写数据，以加快访问速度。

3.2 Cauchy Reed-Solomon Codes

擦除编码通常以 $(k+m)$ 格式指定：k 个数据区块和 m 个奇偶校验区块分布在 $(k+m)$ 个 SSD 上，其中任何 k 个都可以恢复数据。RS 码使用复杂的线性代数运算来生成奇偶校验码。RS 码使用 w 位字来处理数据，并在 GF(2^w) 算法中将其作为介于 0 和 2^w-1 之间的数字进行运算。RS 编码需要矩阵-向量乘法和矩阵求逆运算，而这些运算在 GF(2^w) 中计算量很大。

CRS 码将 RS 码转换为只有 1 比特字的码。因此，GF(2^w) 中的昂贵算术变成了 GF(2) 中的逐位 AND 和 XOR 运算。图 2显示了一个编码示例。通过执行位矩阵-向量乘法，收集所有 $k*w$ 个数据块以生成 $m*w$ 个大小相等的奇偶校验块。这组 $(k+w)*w$ 块被分别传送到不同的磁盘中以实现容错。生成矩阵通常由一个 $k*w \times k*w$ 单位矩阵和一个 $m*w \times k*w$ 子矩阵组成。

3.3 ReRAM 基础

一种金属氧化物 ReRAM 单元，包括顶部金属电极、金属氧化物电阻开关和底部电极[5]。一组 ReRAM 单元可以通过字线和位线形成密集的交叉,这更适合于主存储器。当将矢量电压施加到 ReRAM 交叉开关的字线时，位线末端的电流将表示字线和 ReRAM 单元的点积运算的结果。

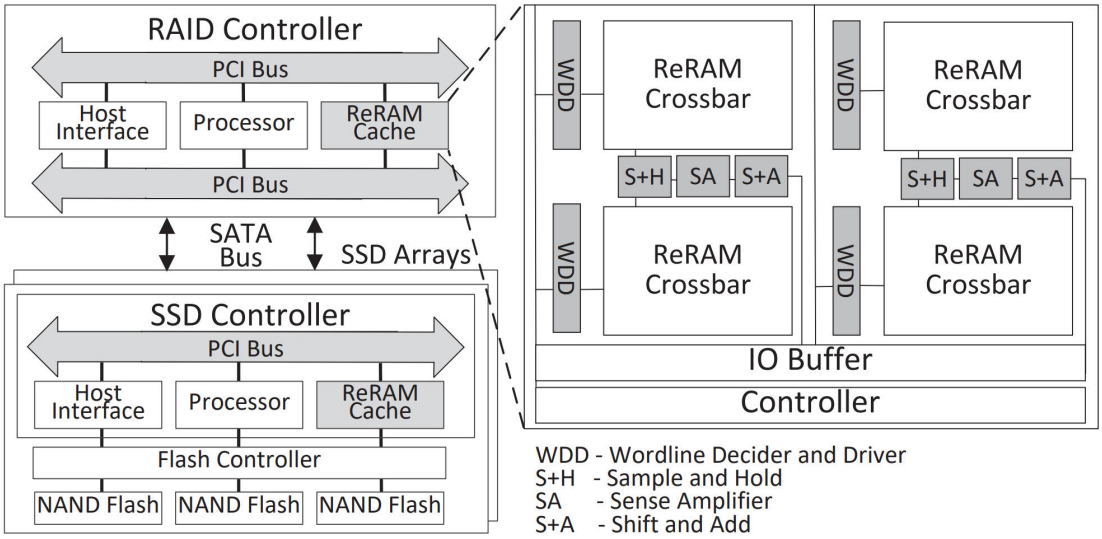


图 3: ReRAID 系统概览

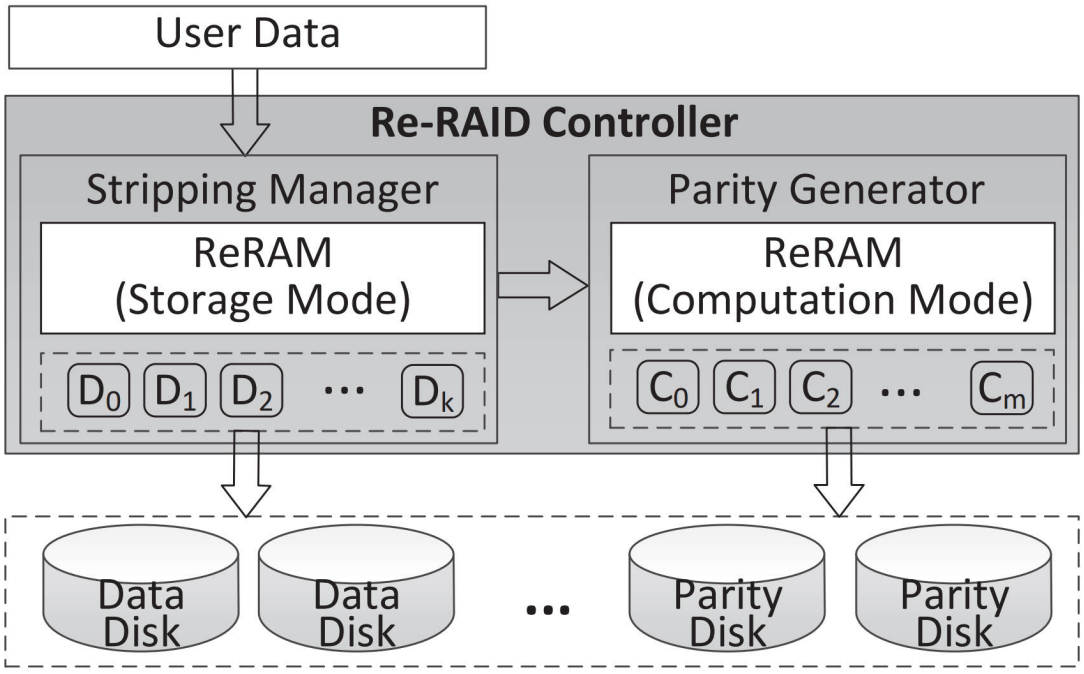


图 4: ReRAID 系统中执行 CRS 编码的数据流

4 本文方法

4.1 本文方法概述

本文提出了 Re-RAID：一种新的基于 SSD 的具有 ReRAM 加速擦除编码的磁盘阵列系统。图 3展示了 Re-RAID 的总体布局，并显示了具有多个数字组件的 ReRAM 的微体系结构。RAID 和 SSD 控制器中的传统 DRAM，即用于保存数据直到可以写入驱动器的高速缓存，在 Re-RAID 中被 ReRAM 取代。Re-RAID 将 ReRAM 划分为两种模式：存储模式和计算模式。处于存储模式的 ReRAM 提供非易失性缓存的功能，而进入计算模式的内存用于对数据进行编码和解码。在编码过程中，将处理器在 RAID 控制器中生成的位矩阵的转置以计算方式映射到 ReRAM。然后，在 ReRAM 交叉开关上执行位矩阵向量乘法后，可以获得条带的奇偶校验码。为了进一步减轻 RAID 控制器的计算工作量，Re-RAID 使用混合的 Cauchy-Vandermonde 矩阵作为编码的生成矩阵。

4.2 Re-RAID 中的 CRS 码

图 4显示了在 RAID 控制器中执行 CRS 编码时的数据流。当用户数据被传送到 RAID 控制器时，两个主要组件，条带管理器和奇偶校验码生成器，通过将数据分成块并计算奇偶校验码来创建条带。被条带管理器分割的条带内的数据块缓存在存储模式下的 ReRAM 上，然后被发送到计算模式下的 ReRAM。将生成矩阵 G^T 的底部 $m*w$ 行（称为 BG^T 矩阵）预先映射到 ReRAM 交叉开关进行计算，并在字线处将数据块转换为 w 位字。然后，可以通过在 ReRAM 交叉开关上执行位矩阵向量乘法来计算奇偶校验码。最后，包括数据部分和奇偶校验部分的条带将被写入到不同的 SSD 驱动器中以实现容错。

4.3 Re-RAID 中的故障重建

- 单个故障。

传统的单个故障恢复方法需要构造译码矩阵，计算复杂度极高，尤其是求逆部分。此外，整个恢复过程是在 RAID 控制器中执行的，如果控制器需要处理其他 I/O 任务，则恢复过程会非常低效。本文提出了一个混合的 Cauchy-Vandermonde 矩阵作为编码的生成矩阵，通过将矩阵 BG^T 的第一行全置为 1 后再用于编码。然后，我们可以将单个故障的重建任务分配给 SSD，每个 SSD 可以在 ReRAM 上执行任务，而不需要解码矩阵。图 5(a) 显示了传统的 Cauchy 矩阵及其 $k*w$ 维度的位矩阵，图 5(b) 显示了我们提出的混合的 Cauchy-Vandermonde 矩阵，它包括 Cauchy 矩阵的 $k-1$ 行，以及可以被视为 Vandermonde 矩阵一部分的 1 行。相应地，位矩阵的前 w 行被填充为 w 维单位矩阵。因此，第一个奇偶性是所有数据块的 XOR 结果。

Re-RAID 使用幸存的数据块和第一个校验块在 ReRAM 上恢复单个故障。RAID 控制器将一个单个故障的重建任务划分为一组子任务，然后将它们交付给幸存的 SSD。一个子任务中幸存的数据块和条带的第一个奇偶校验块被聚集在 SSD 的 ReRAM 交叉开关上。每个块（$k*w$ bit）在交叉开关上占据一行。通过用输入电压 1 激活与条带相关的字线，可以从每个位线（$k*w$ bit）的输出中获得丢失的数据。图 6中的例子很好地说明了这一点。假设图 3中的固态硬盘 D_0 发生故障。为了恢复它，Re-RAID 要求把其他幸存的数据 d_1-d_3 和奇偶校验数据 c_0 放到一个固态硬盘上，比如说，固态硬盘 D_1。每块数据都被映射到 ReRAM 交叉开关上的一行，丢失的数据 $d_{0,0}-d_{0,3}$ 可以通过字线的输入电压 1 来获得。位线中最低的 1 位异或求和结果就是丢失的数据 $d_{0,0}-d_{0,3}$。此外，恢复过程可以用库级和磁盘级的并行方式进行。

- 多个故障。

Re-RAID 保留了多个故障的基本解码规则：解码矩阵乘以幸存数据等于原始数据。一个混合的 Cauchy-Vandermonde 矩阵总是可逆的[10]，所以它可以转换为多个故障的解码矩阵。为了恢复 Re-RAID 中的多个故障的一个可能有效的解决方案是，利用 RAID 控制器中的处理器和 ReRAM 之间的协作。处理器构建一个解码矩阵，然后将其映射到在计算模式下的 ReRAM。通过将幸存数据向量和 ReRAM 上的解码矩阵相乘，可以恢复多个故障。多重故障的解码过程类似于 RAID 控制器中的编码过程。

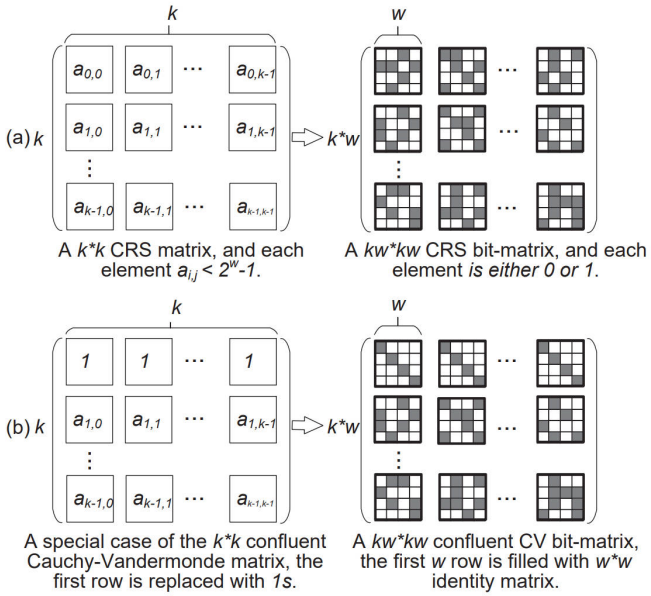


图 5: (a) 传统的 Cauchy 矩阵和它的位矩阵。(b) 本文设计中的优化矩阵

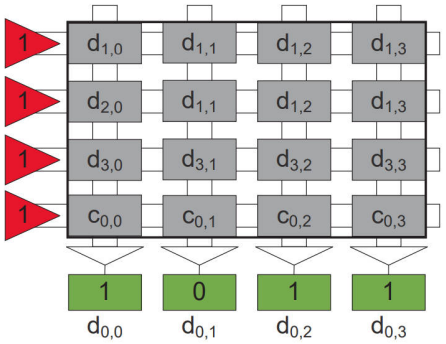


图 6: SSD 中在 ReRAM 交叉开关上进行单个故障重建

5 复现细节

5.1 与已有开源代码对比

本工作参考了两个开源代码库。第一，JErasure 库。这是一个包含多个 Erasure Coding 算法的开源库，在复现过程中参考了其中的编码矩阵生成方法。第二，nvsim 库。这是一个开源的 NVME 模拟器库，本次复现参考了其中对 ReRAM 物理结构的模拟实现。

5.2 复现概述

本次复现完成的主要工作有：

- 编码函数实现。

将生成的编码矩阵映射到 ReRAM Crossbar 后，和输入文件进行运算得到校验码。

- 译码函数实现。

译码函数分为两种：一种是对单个故障进行重建；另一种是对多个故障进行重建。

- 性能计算函数实现。

根据 ReRAM 的时序参数，计算不同操作的时延和次数，依次计算编解码过程中的耗时，最后得到编解码的性能。

- 超大参数下的组合模式编解码。

本功能为复现时新增的技术点，目的是为了增加 Re-RAID 对超大参数进行编解码的能力，本部分将会在 5.4 节详细介绍。

5.3 实验环境搭建

实验的基线利用 JErasure 库[11]在 CPU（i7-12700 2.10GHz）上实现获得。ReRAM 在同样的设备上基于 NVsim[12]实现。其中 ReRAM 设置为 8 个 bank，每个 bank4 个 subarray，每个 subarray 的大小为 514×512。实验结果给出的性能单位都是 MB/Sec。因为实验在 64 位设备上进行，因此 $packet\_size$ 设置为 $8Bytes$，其余参数见表 1。

表 1: 实验参数设置

参数名称	参数值
k	10,20,30,40
m	2,4,6,8
w	8
$packet\_size$	8 Bytes
$buffer\_size$	8 MB

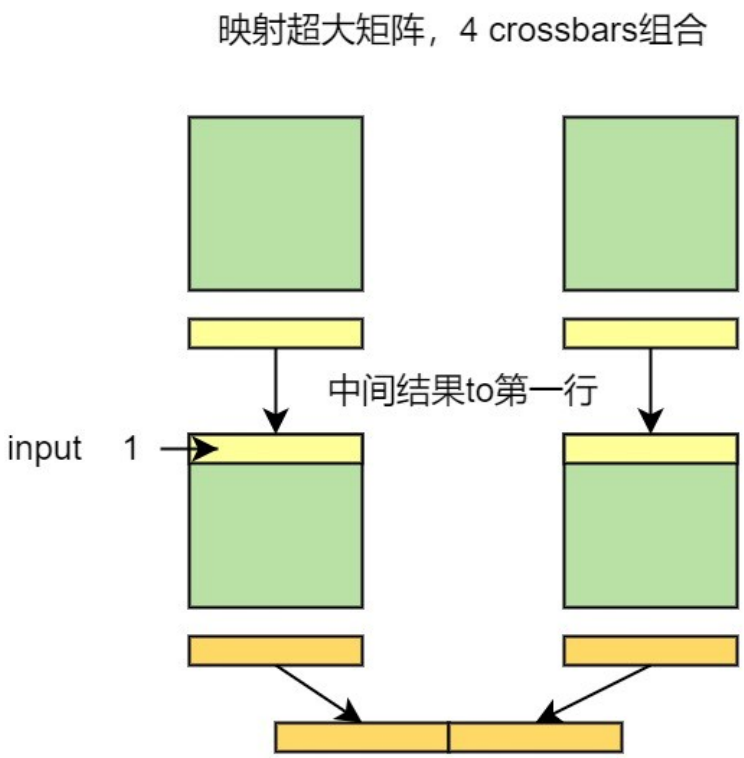


图 7: ReRAM 组合模式下的映射方式

5.4 新增技术点

本次复现过程中，发现 Re-RAID 不能处理大于 crossbar 的超大参数，因此重新设计了 ReRAM 的组合模式，如图所示，当输入的参数大于 crossbar 的行列时，利用多个 crossbar 的组合进行编解码工作。

如 4.2 节中描述的，在编码过程中，需要将大小为 $(k*w)\times(m*w)$ 的编码矩阵映射到 ReRAM Crossbar 中。但是如果用户输入的参数 k,m 太大，导致编码矩阵的规模超过了 ReRAM Crossbar 的规模，则无法对输入数据进行编解码的操作。为了解决这个问题，可以将超大规模的编码矩阵分割为 4 部分，分别映射到 4 个 crossbar（编号为 1,2,3,4）上，然后再进行编码操作，映射方式如图 7所示。为了不影响最终结果的正确性，将编号为 1,2 的 crossbar 的结果映射到编号为 3,4 的 crossbar 的第一行，并在编码时将 1 作为输入，其余编码过程和 4.2 节中类似，此处不再赘述。

6 实验结果分析

6.1 编码性能

首先评估编码性能，具体的结果在图 8中展示。复现的 Re-RAID 相比于基于处理器实现的 JErasure 可以将性能提升 19 到 110 倍。这是因为 Re-RAID 利用了 ReRAM crossbar 内在的计算能力进行基于异或的 CRS 编码，并将数据访问包裹在内存中，从而减少了数据移动的开销。复现工作提升的倍数虽

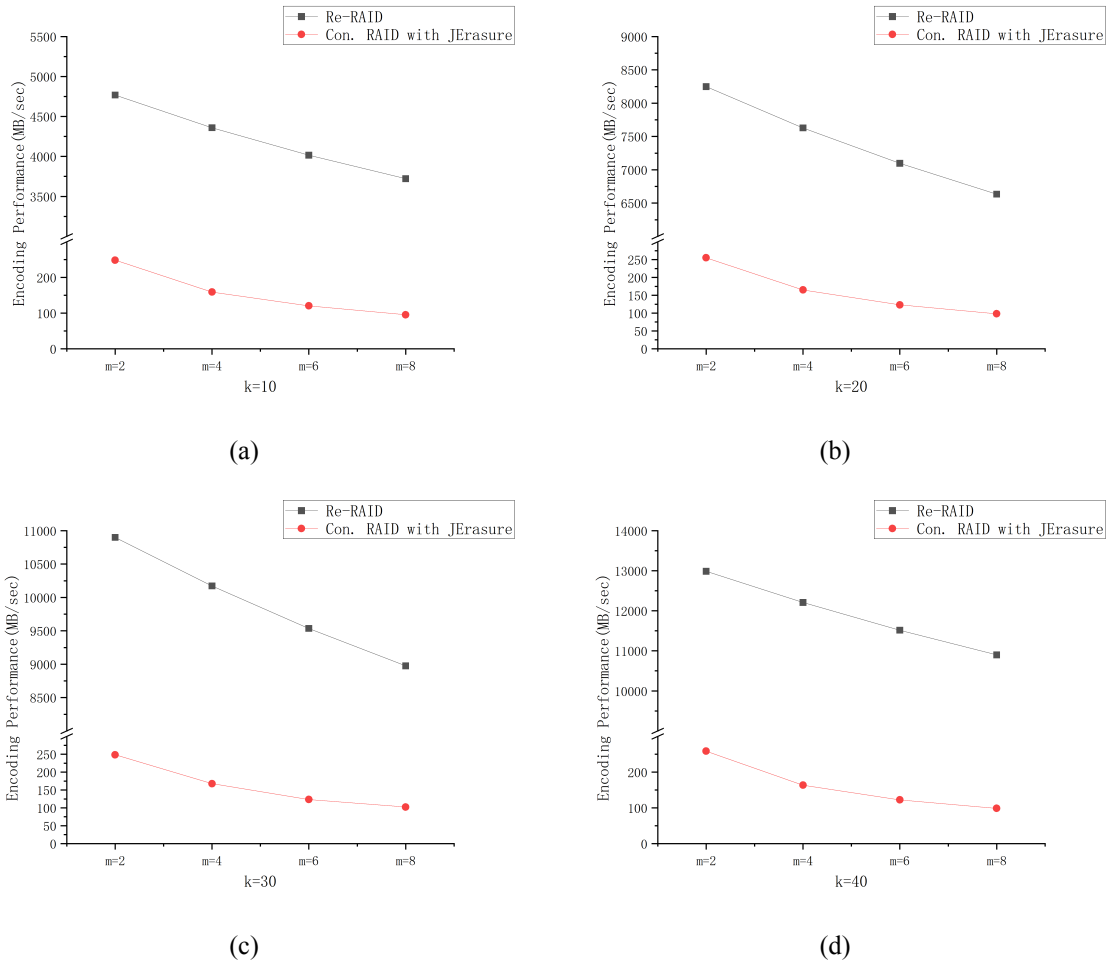


图 8: 编码性能

然与原作有些出入，但是总体趋势类似。另外，在图中可以看到，当 m 固定时，基于处理器实现的编码性能下降了一点。当 m=2 时，性能从 k=10 的 258.7MB/s 下降到 k=40 的 248.6MB/s。然而，如果 k 被固定为 10，性能就会从 m=2 的 258.7MB/s 急剧下降到 m=16 的 95.7MB/s。这是因为随着 m 的增加，对数据块的访问也随之增加，这会给基于处理器的实现带来了更多的性能惩罚。相反，随着 k 的增加，Re-RAID 的编码性能提高了 72%。因为 k 越大，激活字线的时间就越短，所以编码性能就越好。

6.2 单个故障的解码性能

针对 SSD 上的单个故障，RAID 执行重建任务，并将重建任务平均分配给幸存的磁盘。我们用 $k = 10, 20, 30, 40$ 和 $m = 4$ 对文件进行编码，然后使其中一个数据盘出现故障。图 9显示了解码性能结果。我们可以看到，Re-RAID 的性能比基于处理器的实现高出 60.4 倍。这是因为 Re-RAID 在 SSD 中对 ReRAM 内存执行重构任务，并在内存级和磁盘级实现了高并行度。此外，从图中可以观察到，在基于处理器的实现中，解码性能随着 k 的增加而略有下降，而当 k 最大时，Re-RAID 获得了最高的解码性能。这是因为较大的 k 产生具有更多数据块的异或运算，以便在 ReRAM Crossbar 上进行恢复，因此激活字线的数量减少。

6.3 多个故障的解码性能

通过 RAID 控制器中处理器和 ReRAM 存储器之间的协作，RAID 可以恢复多个故障。对于 $k = 10, 20, 30, 40$ 和 $m = 4$ 编码方案，我们使四个数据盘失效。图 10显示了解码性能的比较。我们可以看到，Re-RAID 的解码性能远远高于基于处理器的实现，从 51.6 倍到 149.7 倍。这是因为 Re-RAID 通过将幸存者向量与 ReRAM 存储器上的解码矩阵相乘来恢复丢失的磁盘。

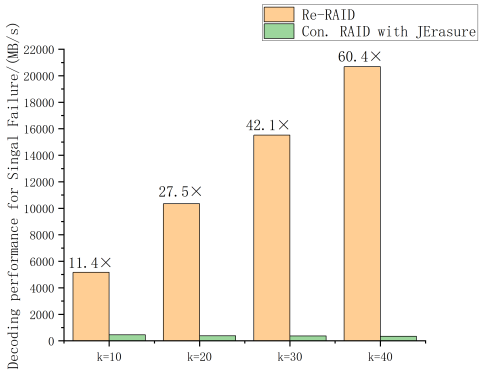


图 9: 单个故障的解码性能

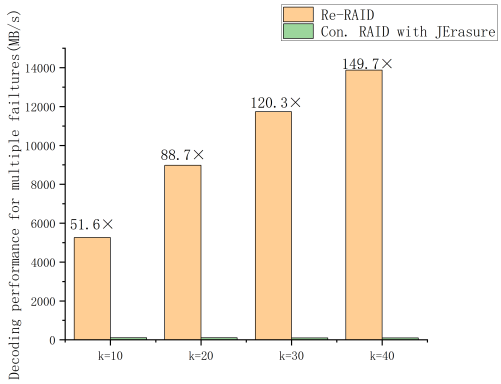
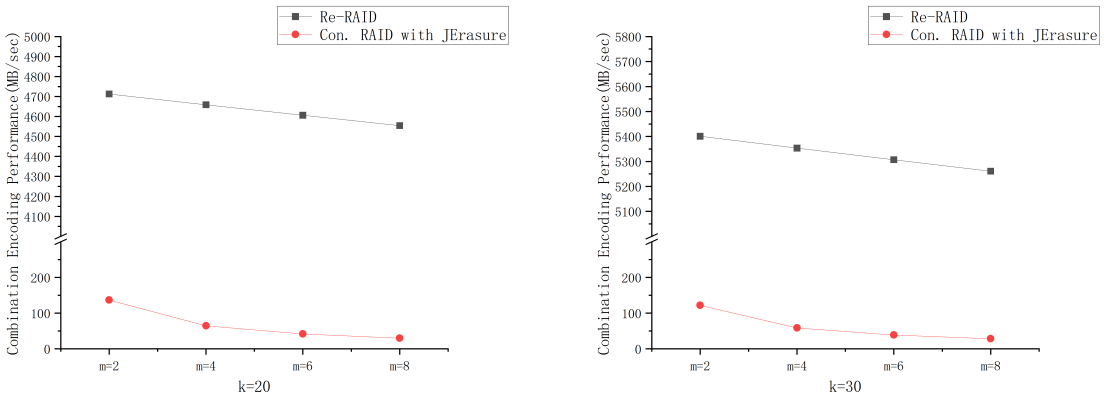


图 10: 多个故障的解码性能



(a) (b)

图 11: 组合模式下的编码性能

6.4 组合模式下的编码性能

为了能够在不改变 Crossbar 的物理结构的前提下对超大参数进行编解码操作，我们使用 4 个 Crossbar 组合对输入文件进行编码。这会导致用于并处处理的 Crossbar 的数量降低从而引起编码性能的下降。但是从图 11可以看出，组合模型下的编码性能仍然远高于基于处理器实现的编码性能，说明利用 Crossbar 组合处理超大参数是行之有效的。

参考文献

[1] DAU H, DUURSMA I M, KIAH H M, et al. Repairing Reed-Solomon codes with multiple erasures[J]. IEEE Transactions on Information Theory, 2018, 64(10): 6567-6582.

[2] DICKSON L E. Linear groups: With an exposition of the Galois field theory: vol. 6[M]. BG Teubner, 1901.

[3] LIU C, WANG Q, CHU X, et al. G-crs: Gpu accelerated cauchy reed-solomon coding[J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(7): 1484-1498.

[4] SATHIAMOORTHY M, ASTERIS M, PAPAILIOPOULOS D, et al. Xoring elephants: Novel erasure codes for big data[J]. arXiv preprint arXiv:1301.3791, 2013.

[5] HAN L, SHEN Z, LIU D, et al. A novel ReRAM-based processing-in-memory architecture for graph traversal[J]. ACM Transactions on Storage (TOS), 2018, 14(1): 1-26.

[6] SONG L, ZHUO Y, QIAN X, et al. GraphR: Accelerating graph processing using ReRAM[C]//2018 IEEE International Symposium on High Performance Computer Architecture (HPCA). 2018: 531-543.

[7] CHI P, LI S, XU C, et al. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory[J]. ACM SIGARCH Computer Architecture News, 2016, 44(3): 27-39.

[8] SONG L, QIAN X, LI H, et al. Pipelayer: A pipelined reram-based accelerator for deep learning[C]// 2017 IEEE international symposium on high performance computer architecture (HPCA). 2017: 541-552.

[9] XIANG L, XU Y, LUI J C, et al. Optimal recovery of single disk failure in RDP code storage systems [J]. ACM SIGMETRICS Performance Evaluation Review, 2010, 38(1): 119-130.

[10] VAVŘÍN Z. Confluent cauchy and cauchy-vandermonde matrices[J]. Linear algebra and its applications, 1997, 258: 271-293.

[11] PLANK J S, SIMMERMAN S, SCHUMAN C D. Jerasure: A library in C/C++ facilitating erasure coding for storage applications-Version 1.2[J]. University of Tennessee, Tech. Rep. CS-08-627, 2008, 23.

[12] DONG X, XU C, XIE Y, et al. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2012, 31(7): 994-1007.