高技术通讯 2018 年 第28 卷 第9 – 10 期:794 ~ 812

doi:10.3772/j.issn.1002-0470.2018.09 – 10.004

优化内存系统能效的 DRAM 架构研究综述①

展旭升②*** 包云岗* 孙凝晖*

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院大学 北京 100049)

摘 要 介绍了不同层次优化内存系统能效研究的现状,对通过修改动态随机存取存储器(DRAM)架构优化内存系统能效的研究进行了详细论述。概述了通过修改内存控制器和操作系统实现的高能效 DRAM 系统的研究。着重介绍了通过修改 DRAM 架构实现内存系统能效优化的研究,并将这些研究分为"低延迟的 DRAM 架构"和"低功耗的 DRAM 架构"两大类进行介绍,其中低延迟架构的研究包括优化关键操作、降低平均访存延迟以及提升请求并发度等 3 个方面;低功耗的架构研究包括细粒度激活、低功耗与低频率芯片、优化写操作、优化刷新操作以及多粒度访存等 5 个方面。最后给出了关于修改 DRAM 架构实现内存能效优化的总结和展望。

关键词 内存,动态随机存取存储器(DRAM),内存控制器,架构,能效,低延迟,低功耗

0 引 言

随着新型数据密集型应用[1](通常需要处理大量的键值存储、图计算、科学计算以及稀疏数据结构等)的发展,新型计算模式[2,3](如内存计算,以新型的硬件和软件系统为基础,将应用的数据尽可能地载入内存,以减少 I/O 操作的并行计算模式)的诞生,以及片上处理器核心数的不断增加[4,5],内存子系统作为整个计算机系统主要组件,其性能和能耗逐渐成为整个系统的瓶颈[6]。

选择内存所采用的存储介质,需要权衡容量、能耗、性能和价格等多个因素。动态随机存取存储器(dynamic random access memory, DRAM)具有高带宽、低延迟、高密度、低成本等特性,已广泛应用于数据中心服务器、嵌入式系统、桌面系统以及移动终端等众多计算系统。特别是对于服务器和数据中心环境,需要低延迟和大容量的内存系统支持,内存系统消耗了超过 25% 的系统能量[7-12],尤其是在有些大内存系统中,DRAM 所占功耗可进一步增长到总功耗的 57%[13]。为了避免 DRAM 系统功耗的过快增加,优化其能效的研究大致从两个方向开展:降低 DRAM 相关操作的延迟,提升系统性能,让应用程序在更短的时间内完成;减少 DRAM 相关操作的功耗,以低的能量开销完成应用程序。对于这两个方面研究,都可以从操作系统级、内存控制器以及 DRAM 等多个层级中的某个单独的层级或者多个层级协同开展。本文侧重于介绍和讨论通过修改 DRAM 结构优化内存系统能效的研究。

当访问 DRAM 芯片时,需要耗费一定的时间将足够的电荷移动到 DRAM 单元或者位线,然后才能将数据从单元中读出来,或者写入单元。为了保证数据正确性,DRAM 制造商需要对芯片设定最小访问时间延迟的限制。与这些访问时间延迟相关的因素有很多,比如,位线的长度、激活操作需要激活的 MAT 数、访问的行是否近期被访问过、访问的行是

① 863 计划(2015AA0153032),国家重点研发计划(2016YFB1000201)和国家自然科学基金(61420106013)资助项目。

② 男,1989 年生,博士生;研究方向:可扩展内存系统结构,多任务 GPU 架构;联系人,E-mail: zhanxusheng@ict.ac.cn

(收稿日期:2018-03-14)

否在行读出放大器中命中、DRAM 单元电荷量以及请求的并发程度等。本文将从优化关键操作、降低平均访存延迟以及提升请求并发度等 3 个方面进行介绍和讨论当前低延迟架构的研究。

DRAM 的功耗主要来源于背景功耗、激活功耗(激活和预充电操作的功耗之和)、读写功耗以及刷新功耗等 4 个方面,优化其中的任何一个或者多个方面都能够节省整体的系统功耗。本文将当前低功耗研究分为细粒度激活、低功耗与低频率芯片、优化写操作、优化刷新操作以及多粒度访存等方面进行分析和介绍。

提升 DRAM 架构的能效,需要权衡硬件开销、容量以及延迟等众多因素。为了能够更好地分析通过优化 DRAM 架构实现系统能效提升的研究现状和未来研究方向,本文主要对近 10 年来的相关研究展开讨论和分析。另外,这些研究是针对 DRAM 内存系统开展的,而不考虑其它类型的主存系统。

本文首先介绍了当前 DRAM 系统的相关背景;其次介绍通过修改内存控制器(memory controller)和操作系统,实现内存系统能效提升的研究现状;然后把通过修改现有 DRAM 架构,优化内存系统能效的当前研究分为"低延迟的 DRAM 架构"和"低功耗的 DRAM 架构"两大类,并分别在第 3 部分和第 4 部分进行详细分析和总结;最后在第 5 部分对修改 DRAM 架构提升系统能效的研究进行总结和展望。

1 DRAM 系统的背景

1.1 DRAM 系统的组织架构

现有 DRAM 是分层式的组织架构,一个晶体管和一个电容器就构成了最小的 DRAM 存储单元,该单元能够存储 1 比特的数据。如图 1 所示[14],整个 DRAM 系统自顶向下可细分为 9 层,分别为通道(channel)、双列直插式内存模块(dual in-line memory modules, DIMMs)、Rank、芯片(chip)、Bank、Subarray、MAT、行(row)以及单元(cell)。最顶层是通道,由一个内存控制器和一个或者多个 DIMMs 组成,其中 DIMMs 间共享命令、地址和数据总线。DIMMs 有独立的双边,每边都有各自的传输信号。每个 DIMMs 包含的 Rank 数为 1 到 4 个不等,内存控制器利用片选信号(select signal, CS)选择将要被访问的 Rank。

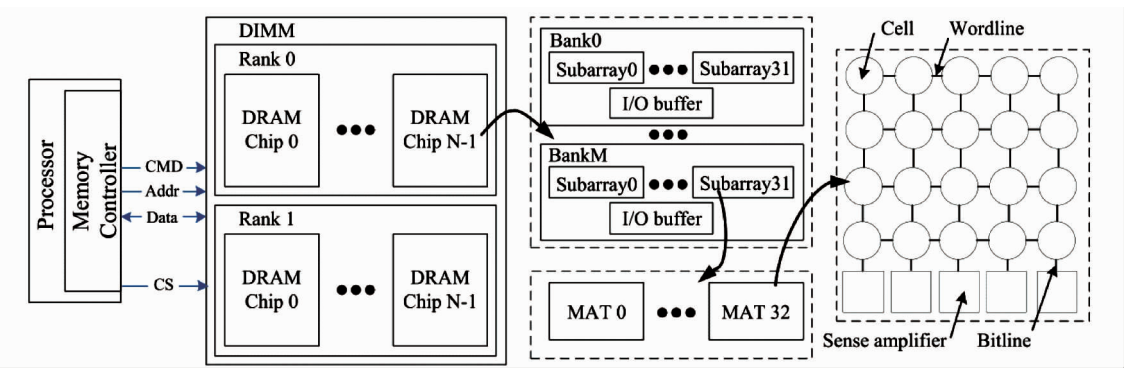

图 1 DRAM 的分层组织架构

联合电子设备工程委员会(Joint Electron Devices Engineering Council, JEDEC)标准将一组能够提供 64 位总线带宽的芯片集合定义为 Rank。由于封装工艺和面积开销,单个 DRAM 芯片的数据输入/输出的引脚个数很有限,通常每次访问只支持 4 位、8 位或者 16 位,相应地分别被称作 x4 DRAM、x8 DRAM 和 x16 DRAM[15]。单个 Rank 中包含的芯片个数等于数据总线带宽除以每个芯片的数据带宽,比如 x4 DRAM 的 Rank 就包含 16 个芯片,而 x16 DRAM 的 Rank 就只包含 4 个芯片。每个芯片内部包含多个 Bank。当前 DRAM 系统支持的并行粒度可分为通道级并行、Rank 级并行以及 Bank 级并行,其中,通道级并行属于完全并行,访问不同通道的请求可以完全并行处理;而剩余的 2 种并行属于部分并行,因为它们需要额外的时间开销用于在 Rank 或者 Bank 之间切换。Bank 是可并行处理的最小组

织形式,访问相同 Bank 的请求只能顺序执行。

图 2 以 2 Gb x8 DDR3-1600 芯片为例[10],进一步显示了 Bank 内部的细粒度组织形式。实际上 Bank 可以简单地设计为由很长的字线(wordline)和位线(bitline)组成的大阵列。其中,字线连接多个位于同一行的 DRAM 单元组成行,而位线连接位于同一列的多个 DRAM 单元和 1 个读出放大器。由于这种设计会带来很高的访问延迟和能量开销[10,12,16],因此每个 Bank 被进一步划分为多个更细粒度的结构 Subarray。相同 Bank 中的 Subarray 共享相同的全局资源,如 I/O buffer 和全局行解码器(row predecoder)。全局行解码器确定当前 Bank 中被选定的全局字线。每个 Subarray 可被进一步水平切分为多个 MAT。每个 MAT 包含 4 个重要组件:1 个局部行解码器、1 个专用 HFF (helper flip-flop),1 个读出放大器和 1 个 512 × 512 的 DRAM 单元阵列。当 1 个行地址到达 Bank 的全局行解码器,1 个 Subarray 将会被选定,并且该地址也会通过全局字线被发送到 Subarray 的每个 MAT。每个 Subarray 中的所有 MAT 同时操作,将目标行中的数据载入到读出放大器。真正被操作的数不是整行数据,而是每个 MAT 只提供 4 比特。MAT 中的 HFF 用于锁存选中列的数据,并将这些数据发送到外部数据总线。

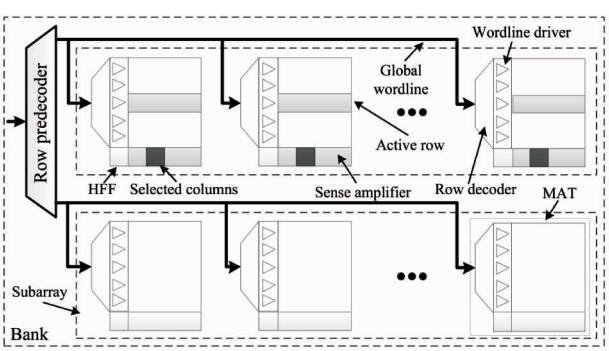

图 2　Bank 内组织结构

1.2　基本操作与时间参数

当前内存系统支持 3 类操作:读、写和刷新。其中,读操作和写操作用于 DRAM 系统和 CPU 之间交换数据,刷新操作用于维持数据的正确性。之所以需要执行刷新操作,是因为 DRAM 单元存在电荷泄露的现象,所以需要定期地对每个单元充电,以便维护每个单元所存储数据的正确性。由于上述 3 类操作都不属于原子命令,当内存控制器接到这些命令会进一步翻译成新的由激活、读、写和预充电等名称组成的命令。激活命令(activate command, ACT)执行的操作时先将被访问 DRAM 行的所有单元的数据载入到读出放大器,然后恢复这些单元中的数据。读命令(read, RD)将读出放大器中的数据借助数据总线驱动到内存控制器。写命令(write, WR)执行的操作与 RD 命令类似,不过是将数据从内存控制器返回到目标读出放大器。预充电命令(pre-charge, PRE)重置读出放大器和位线,以便接收下次对该 Bank 的访问。

图 3 解释了不同 DRAM 操作的命令组合和相关的时间参数。每条 DRAM 操作命令会与一个或多个时间参数相关。ACT 命令需要花费 tRCD(row-column delay)的时间将选中的整行数据载入读出放大器,然后花费 tRAS(row access strobe)的时间完成该行的数据恢复过程。现有 DRAM 系统利用短的连续的突发过程在芯片间传输数据,tTBURST 指示一条读或者写命令将数据突发传输到数据总线的时间。tRP(row pre-charge timing)表示从发出 PRE 命令到目标 Bank 完成预充电过程的时间。tRC 表示连续访问同一个 Bank 的不同行的最小时间间隔,等于 tRAS 和 tRP 的总和。tCWD 是从命令总线收到写命令开始,直到内存控制器把数据移动到数据总线之间的时间。tWR 是读出放大器将数据恢复到目标行的最短时间。刷新操作关联紧密的是 tRFC 和 tREFI,其中当一个 Rank 执行刷新命令后,需要等待 tRFC 时间处于不处理其他内存请求的状态;tREFI 表示两条刷新命令之间的最小时间间隔。

1.3　读出放大器管理策略

DRAM 系统针对读出放大器的管理策略分为两大类:开页策略和闭页策略。对于开页策略,当某个行被激活和访问后,一直处于打开状态,以便下次访问该行,这样就可以减少相应的激活操作,从而节省功耗,但是如果下次访问该 Bank 的不同行,需要关闭(预充电)当前行,然后再激活将要访问的行,会导致访问延迟的增加。闭页策略的设计更加简化操作,每次对激活的行执行相应的读或者写操作后,紧

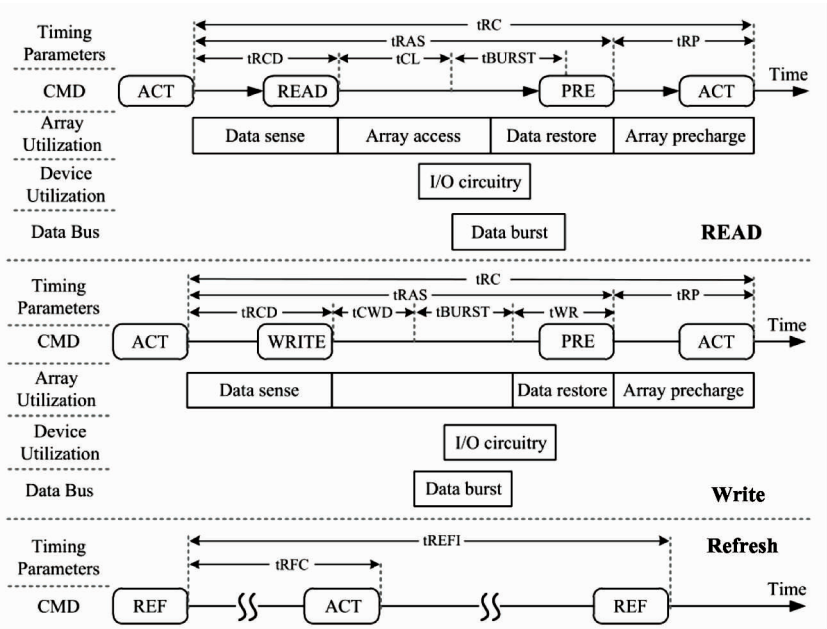

图 3　Bank 操作相关的时间参数

接着预充电当前访问的行,从而可以避免下次访问不同行时,造成的额外时间延迟。

1.4　系统功耗类型

DRAM 系统的功耗主要来源于多个方面,可以归结为 4 大类:背景功耗、激活功耗(激活和预充电操作的功耗之和)、读写功耗以及刷新功耗。背景功耗主要指外围逻辑和延迟锁定环(delay locked loop, DLL)[17,18],这部分功耗是一直存在的,但是不同功耗模式的背景功耗也存在差异。激活功耗是指激活命令和预充电打开和关闭相应 DRAM 行所消耗的功率,因为激活操作和预充电操作总是成对执行的,所以分到了同一组;读写功耗是读写命令将数据读出或者写入读出放大器的功耗。刷新功耗是指为了维持数据完整性刷新数据阵列所消耗的能量。

由于背景功耗是 DRAM 系统运行过程中一直存在的,为了减少这部分的能耗,当前 DRAM 系统支持多种低功耗的模式,一般在当前 Rank 处于无请求的空闲状态时,DRAM 芯片会切换到这些状态。但是这些低功耗模式的粒度很粗,对于访存比较密集的应用,芯片基本上就不会进入低功耗的状态。

现有 DDR3 DRAM 芯片提供待机、休眠和自刷新(self refresh, SR)等 3 种不同的状态,并支持 7 种功耗模式,多种模式间的转换如图 4 所示,不同的功耗模式直接决定了背景功耗的大小。图中 CK 是 Clock 缩写,ODT 是 on-die termination 的缩写。当 DRAM 芯片处于主动待机模式时,可以立即执行接收到的 DRAM 操作,同时背景功耗也最高。芯片只有处于待机模式才能执行访存操作。对于主动待机模式的 DRAM 芯片,如果将时钟使能信号(clock enable,CKE)设置为低电平,则 Rank 进入主动休眠模式;如果 Rank 中的所有 Bank 都没有被访问,且没有打开的行,则可将 Rank 设置为预充电待机模式。如果将DLL也设置为无效状态,Rank将进入慢速

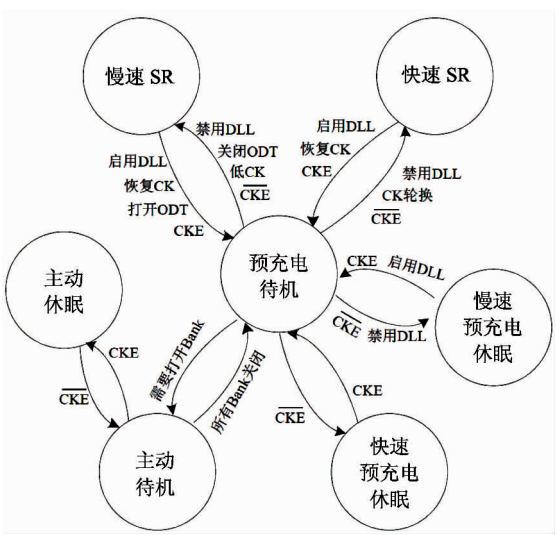

图 4　背景功耗模式间的转换关系

预充电休眠模式。慢速预充电休眠模式比快速预充电休眠模式消耗更少的功率,但需要更长的时间返回到待机模式处理后续的访存请求。当Rank没有访问请求时,可以将芯片设置为背景功耗最低的(快/慢速)自刷新模式,该模式执行周期性的刷新操作,以消耗最少的功率维护数据完整性。休眠模式能够节省大量的背景功耗,但是退出该模式所引入的退出延迟很高,比休眠模式的退出延迟高 2 到 3 个数量级。

与 DDR3 相比,DDR4 将设备核心电源 *VDD* 从 1.5 V降低到1.2 V,该设计很大程度上降低了 DRAM 系统的背景功耗。另外,DDR4 新增加 CAL (command address latency)特征[19],该特征在默认情况下会关闭芯片的 I/O 缓冲区,只有当地址和命令被锁存时,命令和地址接收器才能被激活。

DDR3 提供 4 个模式寄存器(mode register, MR),允许内存控制器通过模式寄存器设置(mode register set, MRS)或模式寄存器读取(mode register read, MRR)命令来写入或读取这些模式寄存器,方便用户自定义不同的功能和模式。默认情况下 MR 是未被初始化的,需要通过上电或复位操作将其进行初始化或者重新初始化。DDR4 DRAM 芯片扩展已有寄存器的访问接口。可编程功能 tCAL 指定模式寄存器 MR4 [A8:A6]定义片选(CS _ n)和 CMD / ADDR 之间的时钟延迟数。CAL 增加了命令的延迟,但是可以让芯片更长时间地保持在低功耗状态。其它低功耗模式仅对退出当前功耗模式后的第一条命令增加很大的延迟,而进入 CAL 模式是对所有的命令都增加了延迟。

2 优化内存系统能效的现状

本小节主要介绍当前通过修改编译器、操作系统或内存控制器实现内存能效优化的研究方案。

2.1 低功耗状态转换方案

已有许多研究在内存控制器中实现低功耗状态间的转换策略,Delaluz 等人[20]提出一种基于编译器的优化框架,将应用程序中具有相似生命周期的数据放置到相同的一个或一组 Bank 中,当数据的生命周期结束时,该 Bank 或该组 Bank 切换到低功耗状态。基于这种软件的编译方法,Delaluz 等人[21]进一步提出软硬件结合的节能策略。利用编译分析的软件方法在程序代码中自动插入模式转换指令。提出自我监控的硬件方法,使用不同的启发式策略来预测访存的空闲时间段,并根据该时间段 DRAM 芯片切换到低功耗模式。Fan 等人[22]提出了一个分析模型,利用指数分布近似 DRAM 芯片的空闲时间,并通过对典型的基准测试进行分析发现,简单策略(DRAM 芯片空闲时,直接切换到低功耗状态)性能优于复杂的策略(预测 DRAM 芯片的空闲时间)。RAMZzz[23]利用内存控制器监控访存的局部性,将局部性相似的页面划分到相同的 Rank,然后利用动态页面迁移的方法将 Rank 间短的空闲时间合并成较长的空闲时间,并提出一个预测模型用于估计适合不同优化目标的降级时间,以便准确控制功耗状态转换。

由于大多数的文件 I/O 访问需要通过系统调用完成,并且操作系统知道这些访问的完成时间。Bi 等人[24]利用 I/O 处理例程能够将空闲的 Rank 切换到低功耗模式,并提出预测算法,预测可能被访问的 Rank,并在系统调用的入口处打开该 Rank,减少 DRAM 低功耗状态转换的延迟。Hur 等人[25]提出一种估计节流延迟(内存控制器中阻塞访存命令的时间段)的方法。首先通过实验生成 3 个描述 DRAM 状态的关键参数,然后利用离线模型生成器(只在系统配置时被训练 1 次)来确定每个参数的特定系数。从而可以通过少量的动态和逻辑来计算节流延迟。Amin 等人[26]提出 2 种利用 DRAM 低功耗状态提升能效的策略:RARE (Rank-aware replacement)是一种末级缓存的替换策略,防止替换映射到优先处理的 Rank 的块,从而减少对该 Rank 的整体访问;RAWB(Rank-aware write buffer)缓存特定芯片的写请求,在 DRAM 芯片处于全功率模式时批量发送这些写请求,从而延长潜在的空闲时间。

2.2 请求调度方案

内存控制器接收来自处理器端末级缓存缺失和写回的请求,并缓存到其中的内存事务队列中,然后发出适当的 DRAM 命令来逐个完成这些请求。为

了保持芯片的正确性,内存控制器需要监控每个 Bank、每条总线以及每条访存请求的状态。DRAM 访存命令的顺序和交错方式会影响整个 DRAM 系统的吞吐量和访存延迟[13]。近年来,处理器和主存之间的速度差异依旧在不断增加,提升系统性能也一直是内存控制器的首要任务。不过,多核处理器和数据中心的发展,内存带宽作为多核系统中的共享资源,能否有效使用它对 DRAM 系统的吞吐量和公平性至关重要。

内存控制器灵活地调度不同通道和 Bank 间的请求,严格遵循 DRAM 芯片的时序约束,同时,有效地利用请求之间潜在的并发性。FR-FCFS[27] 优先处理 DRAM 行命中的请求,如果所有请求都不满足行命中的条件,则按照先来先服务的方式进行处理。该算法在单线程系统中能够实现最好的性能,但是在多核系统中会造成饥饿等问题。STFM[28] 区分处理来自不同线程的请求,提出公平 QoS(quality of service)度量,对于两个优先级相同的应用同时运行时,相比于它们单独运行的情况,它们各自的性能下降比例相似。STFM 会优先化性能下降最多的线程。PAR-BS[29] 为了避免饥饿,维持请求的公平性,按批发射 DRAM 请求,并能够并行处理来自相同线程的请求。ATLAS[30] 会定期重新设定线程的优先级,将过去一段时间获得服务时间最少的线程的优先级设置为最高。ATLAS 能够提升系统的吞吐量,但是会损害访存密集型应用的公平性。TCM[31] 基于线程的访存行为,预先将线程分为延迟敏感型和带宽敏感型两大类,对不同的分类应用不同的调度算法。Ipek 等人[32] 利用强化学习的方法观察系统的状态来估计采用不同的动作对系统性能的长期影响,并不断地优化基于估计值的调度策略,让内存控制器能够适应负载的动态行为特征。该策略适用于访存密集型负载自动实现总线利用率和吞吐量最优。MORSE[33] 基于 Ipek 等人的方法,提出了一种目标可配置的自动最优调度策略,实现对性能、吞吐量、能耗以及公平性等不同目标的优化策略。

2.3 刷新调度方案

DRAM 控制器支持基本的延迟刷新的调度策略,该策略会优先处理访存队列中的命令,只有当访存命令队列为空,或者刷新命令已经被延迟了 8 个 tREFI 时,才会执行刷新命令。Elastic refresh 机制[34] 优化上述方案,当访存队列为空后,等待一段时间,如果这段时间没有新请求到达,则发送刷新命令,否则执行访存请求。Coordinated refresh 方案[35] 是对 Elastic refresh 方案的提升,内存控制器不需要发射所有挂起的刷新命令,而是将 DRAM 切换到 SR 模式下处理这些挂起的刷新命令,从而节约背景功耗,并减轻刷新操作对系统性能的影响。Mukundan 等人[36] 在高密度 DRAM 系统中发现一个普遍现象:当内存控制器发送刷新命令到 1 个 Rank,它的命令队列将被该 Rank 的相关命令占用,所以无法处理其他 Rank 的请求。他们提出 DCE(delayed command expansion)方案来限制当前正在刷新的 Rank 的访存请求数量,又提出 PCD(preemptive command drain)方案来优先化未被刷新 Rank 的访存请求。Nair 等人[37] 提出了一种可中断刷新操作的刷新暂停方案,当 DRAM 系统正在执行刷新操作时,如果有新的访存请求到达,则 DRAM 系统会暂停正在进行的刷新操作,转而处理新到达的请求,请求完成后恢复先前的刷新操作。

2.4 内存控制器级刷新减少方案

对 DRAM 行执行一次读/写操作类似于对该行执行一次刷新操作。Song[38] 设计了一种选择性的 DRAM 刷新方案,为每个 DRAM 行关联一个有效位,记录该行在当前刷新周期内是否被访问过,如果被访问过,则跳过对该行的刷新,并重置相应的位。Smart refresh[39] 与该机制类似,为每个 DRAM 行提供一个超时计数器,当某行被执行读/写/刷新操作时,复位相应的计时器。刷新某个 DRAM 行时,如果计数器还没有超时,则跳过刷新该行,否则执行刷新。

当前制造工艺使得 DRAM 单元的数据维持能力不同,小部分 DRAM 单元具有较弱的数据维持能力。RAIDR[40] 根据 DRAM 单元保留时间的不同,采用各种刷新率刷新具有不同保留时间的行,减少了不必要的刷新操作。DTail[41] 以较低的存储成本,将不同 DRAM 行的保留时间存储到 DRAM 内,并通过修改 DDRx 协议添加一个新的命令“silent re-

fresh”,该命令只增加行地址计数器的值,而不是执行真正的刷新操作。REFLEX 方案[42]利用 DDR4 的 4x 刷新粒度减少每次 AR 刷新操作所包含的行数,并提出与“silent refresh”功能类似的“dummy refresh”命令,该命令只增加 DRAM 设备中的刷新计数器值,并不执行刷新操作。近期研究发现,DRAM 单元存在 VRT(variable retention time)现象[43],即部分 DRAM 单元的维持时间会发生变化。Khan 等人[44]发现将 DRAM 单元探测技术与 SECDEC-ECC 和防护带技术相结合,能够有效地避免 VRT 引起的相关错误。

2.5 操作系统级刷新减少方案

ESKIMO[45]基于程序的语义信息,由操作系统的内存管理器分配或释放内存空间。它利用语义指示内存区域的有效或无效状态,从而避免刷新已被释放的行。Flikker[46]借助程序开发人员将应用程序的数据划分为关键和非关键两部分。程序运行时,操作系统会将不同类型的数据分配到不同的内存区域。关键数据所在的内存区域按正常刷新率刷新,其他内存区域的刷新速率相对较低。RAPID[47]通过修改操作系统,可以获取每个物理页面的保留时间。基于保留时间,操作系统可以优先分配保留时间较长的页面。DRAM 芯片按已分配页面的最低刷新周期刷新已分配的页面而不是所有的内存页面。随着页面使用数量的增加,其有效性下降。基于相同的观察,RIO[48]利用页面保留时间信息来避免使用保留时间相对较短的页面。但 RIO 会导致物理地址的不连续性,并损害大页面和 DMA 的性能。Jagadish 等人[49]提出软硬件协同机制,将硬件的地址映射方案和 Bank 级的刷新调度方式都暴露给操作系统,操作系统使用刷新感知的进程调度算法,在每个任务的给定调度时间范围内,只刷新预期不会收到访存请求的 Bank。

2.6 电荷量优化方案

电荷泄露现象会导致 DRAM 单元中的电荷量随时间逐渐减少,可以通过读、写或者刷新操作来补充 DRAM 单元中的电荷,使其恢复到指定的阈值。DRAM 行中单元电荷量的多少决定了访问该行所需要的时延。所访问 DRAM 行的电荷量越高,就越容易恢复到指定的阈值,访问延迟也就越小;相反,如果所访问行很久没有被访问或者刷新,那么这些单元的电荷量就会很低,需要补充更多的电荷才能恢复到阈值,所以访问延迟也就越大。由于近期被刷新的行具有高的电荷量,NUAT[50]提出访问时间可变的内存控制器,以较低延迟访问近期刷新过的 DRAM 行。ChargeCache[51]利用近期被访问的 DRAM 行具有更多电荷的特性,在内存控制器中新增一张表,用于记录最近被访问行的地址,如果后续的访存请求在该表中命中,则以较低的延迟访问该行。随着时间的推移,动态地更新该表中的内容。Restore truncation[52]对被访问的行执行恢复操作时,不需要将该行的电荷完全恢复到指定的阈值,而是恢复到下次刷新之前不至于丢失该行数据的电荷值,从而节省激活功耗。

3 低延迟的 DRAM 架构

3.1 优化关键操作

许多应用程序和操作系统中都存在批量数据移动的关键操作,但是对批量数据移动的操作执行效率很低,因为需要将数据从 DRAM 传输到处理器,然后再通过狭窄的芯片外通道回传到 DRAM,仅使用这个狭窄通道进行批量数据移动会导致高延迟和能耗。DRAM 内有限的连接线路仅允许在单个 DRAM 子阵列内部快速地实现数据移动,每个子阵列只有几兆字节大小,极大地限制了 DRAM 内快速批量数据移动的范围。

RowClone[53]是直接在 DRAM 内部执行批量复制和初始化任务的机制,不需要通过内存通道传输任何数据来完成此类操作。其中,块数据拷贝是指从物理内存的一个位置拷贝大量的数据到另外一个位置;块数据初始化是指将大量的数据初始化为一个特定的值。RowClone 实现了 2 个机制:对于共享读出放大器的行,利用 2 个连续的激活命令,将数据从源 DRAM 行复制到读出放大器,然后从读出放大器复制到目的行;为了在没有共享读出放大器的行之间有效地复制数据,它使用 DRAM 芯片的内部共享总线在 2 个 Bank 之间快速复制数据。

LISA(low-cost inter-linked Subarrays)[54]在邻接的 2 个 Subarray 之间添加低成本的连接线,并用这些连接线互联邻接 Subarray 的内部线(位线),以便 LISA 能够以低的面积开销实现横跨多个 Subarray 以较宽的带宽进行数据传输。LISA 基于 2 个重要发现:当访问 DRAM 中的数据时,需要将整行 DRAM 单元的行通过子阵列的位线传输到读出放大器;相同 Bank 中部署不同的 Subarray 在物理位置上挨得很近。LISA 基于新的 DRAM 组织结构衍生出一个新的 DRAM 命令,被称作 RBM(row buffer movement)。当一个 Subarray 的某行数据被锁存在读出放大器时,RBM 可以将这些数据移动到另一个 Subarray 中的不活动的读出放大器中,而不需要在 DRAM 中通过狭窄的内部数据总线。

Ambit[55]是一个用于批量位操作的内存加速器,将批量按位操作直接集成到 DRAM 存储器阵列中,完全在 DRAM 内部完成按位操作。该研究的初衷是当前系统中批量按位操作在内存通道传输大量的数据,会导致延迟变高,带宽和能量浪费。Ambit 通过同时激活共享同组读出放大器的 3 个 DRAM 行,使 DRAM 系统实现按位 AND 和 OR 操作。另外,通过对读出放大器进行适度的改变,DRAM 系统可以使用读出放大器内的反相器执行按位 NOT 操作。基于上述的 2 个改动,结合指令集和系统的支持,Ambit 可以在 DRAM 内部高效地执行任何批量按位操作。

3.2 降低平均访存延迟

CHARM[56]是减少平均主存访问延迟的非对称 DRAM 架构。该研究分析现有 DRAM 设备的访问时间和周期时间,发现降低 MATs 内的数据通路电容和缩短从 I/O 到 MATs 的数据传输距离是降低延迟的关键。通过减少每个 MAT 内字线条数来减少该 MAT 的周期时间,同时以面积增加为代价降低了存取时间和功耗。由于许多应用程序对 DRAM 的访问不是均匀分布在它们所覆盖的内存区域,CHARM 将具有不同纵横比的 MATs 放置在一起来减轻面积开销,并将高纵横比的 MATs 放置到接近 I/O 的物理位置来进一步减少部分芯片的访存时间。重组列解码器和 Bank 间的数据线,使得中心高纵横比 MATs 的访问时间大约是其他正常 MATs 的访问时间的一半,并将经常访问的页面分配到这些部分。

DAS-DRAM[57]是混合位线的 DRAM 架构,在不对称的 DRAM 中执行快速 Subarray 和慢速 Subarray 之间的直接 DRAM 行迁移。混合位线 DRAM 芯片具有不均匀的子阵列,比如包含 2 种类型的 Subarray:一类是访问延迟时间较短,但密度较小的快速 Subarray;另一类是访问时延长,但是容量很大的慢速 Subarray。基于 2 种阵列的异构特性,大容量的慢速 Subarray 可以分摊快速 Subarray 的芯片面积开销。为了最大化快速 Subarray 的利用率,为每条位线新增加一个“迁移单元”。这个单元有 2 个共享 1 个存储电容的访问端口,也允许 2 个未关联的位线访问共享存储单元,主要用于行数据传输的临时存储。大致的操作过程是将源 DRAM 行复制到临时迁移行,然后从临时迁移行传输到目标行。

TL-DRAM(tiered-latency DRAM)[16]利用隔离晶体管将 DRAM 中原有的每条长位线分成较短的两段,直接与行缓冲区连接的一段被称为“近段”,另外一段被成为“远段”,以短位线 DRAM 的延迟访问近段的数据,而不会产生高的每比特成本。因为 DRAM 的阵列中,位线的长度相对比较长,所以会产生高的寄生电容,并且该电容是 DRAM 延迟的主要来源。特定的低延迟 DRAM 会使用更短的位线和更少的单元,但是由于更大的读出放大器面积开销,因此具有更高的每比特成本。TL-DRAM 提出了使用低延迟段作为硬件管理或软件管理缓存的机制。其中,为了将近段部分作为一个操作系统透明的硬件高速缓存来管理,TL-DRAM 提出了一个与 RowClone 类似的 Subarray 之间数据移动方案。

Son 等人[58]在读出放大器和位线之间增加隔离晶体管,将位线与读出放大器进行分离,实现位线在执行预充电操作时,读出放大器中缓存的数据依旧有效。该研究的出发点是读出放大器的管理策略会对 DRAM 的延迟有很大的影响,无论是采用开页策略还是闭页策略,不同应用的性能也有很大的差异,所以他们提出的这种新的 DRAM 架构能够将这两种策略的优势结合,避免通过预测的方式来选择

读出放大器的管理策略。毕竟内存控制器缺乏关于后续 DRAM 请求的相关信息,仅根据已有特征很难准确地预测程序的后续访存行为。他们提出的这种通过去耦合行缓冲区的实现的预充电策略,允许位线在大多数行缓冲区未命中时对关键路径进行预充电,从而实现在行缓冲区缺失的情况下,只需要去激活行缓冲区,该操作所需的时间小于 1ns(因为去激活行的大部分时间花费在预充电与行缓冲区连接的位线)。

MCR (multiple clone row) DRAM[59] 将多个物理行组织成一个逻辑上的 DRAM 行,从而使多行具有相同的数据。MCR 中包含的多行是被同时打开或关闭的。基于该特性,MCR 通过增加读出 DRAM 单元的数量来加速读出过程,而不引入新的面积开销。MCR 中的 DRAM 单元在没有额外刷新命令的情况下能够更加频繁地执行刷新操作,DRAM 单元具有更短的更新间隔,DRAM 单元的电荷泄露也越少。随着泄露的电荷量减少,预充电命令能够在被激活的单元完全恢复之前被提供服务(即提前预充电)并且在刷新单元被完全恢复之前完成刷新操作(快速刷新)。由于 MCR 是多个普通行的逻辑分组,所以用户可以改变 MCR 中包含的行数、MCR 在总的 DRAM 中所占的比例等。

3.3 提升请求并发度

SALP-2(Subarray-level-parallelism-2)[60] 通过观察发现 tWR(Bank 收到写命令后,需要耗时 tWR,让读出放大器将位线驱动到指定的电压。位线达到指定电压之前,不能对 Bank 执行预充电操作,否则不能保证数据被正确存入 DRAM 单元)是 Subarray 的内部功耗时,从而在当前激活的 Subarray 发送预充电命令之前,向另外一个 Subarray 发送激活命令。SALP-2 将当前激活 Subarray 的 tWR 时间与另一个子阵列的激活时间重叠,从而缩短了请求服务时间。现有 DRAM 中,相同 Bank 中的所有 Subarray 共享行地址锁存器,SALP-2 为了能够让两个子阵列同时保持激活状态,每个 Subarray 的外围逻辑都增加一个小的锁存器。

MASA(multitude of activated Subarrays)[60] 利用每个 Subarray 都有独立的局部读出放大器结构,可以将近期访问的数据缓存到其所在 Subarray 的读出放大器中。MASA 在 SALP-2 的基础上为每个 Subarray 增加 1 位锁存器,用于接收来自内存控制器的选定信号。为了访问指定激活 Subarray 的数据,内存控制器设置指定 Subarray 的选定信号,同时清除同 Bank 中其它 Subarray 的选定信号。MASA 减少 Subarray 之间的硬件资源共享,使得内存控制器能够并发激活多个 Subarray,减少 Bank 内请求序列化。同时让多个 Subarray 的读出放大器处于活动状态,从而提升读出放大器的访问命中率。

SARP(Subarray access refresh parallelization)[61] 修改现有 DRAM 组织结构,允许 Bank 对一些 Subarray 执行刷新操作时,同时能够为其它没有执行刷新操作的 Subarray 提供内存访问。因为对 Bank 执行刷新操作时,只有少数的 Subarray 真正在执行刷新操作,而其他的 Subarray 和 I/O 总线处于空闲状态。由于每个 Subarray 都有一组读出放大器,且不与同 Bank 的其它 Subarray 共享。SARP 利用每个 Subarray 的读出放大器完成该 Subarray 的刷新操作,而不需要借助 I/O 总线传输任何数据。

CREAM[62] 与 SARP 相同的发现:DRAM 的单个 Bank 执行刷新操作期间数据移动的范围仅限于 Subarray 和相应的读出放大器之间,也不会引起与其它 Subarray 内存访问的资源争用。CREAM 提出 2 种方案:SRLR(sub-rank-level refresh) 减少同时刷新的 Bank 个数,把节省的功耗用于内存访问以便提升性能;SALR(Subarray-level refresh) 将 8 个 Bank 组成的初始 Rank 可以进一步划分为 2 个、4 个或 8 个 sub-rank,每个 sub-rank 分别允许 4 个,2 个或 1 个 Bank 同时执行刷新操作。当执行刷新操作的 Bank 数越少,CREAM 越能利用节省的电能并行地执行读/写访存操作。

3.4 小结

表 1 列出本节关于低延迟 DRAM 架构的研究。关键操作是指操作系统或者应用程序中对系统延迟影响最大的操作。当前针对系统级操作的优化主要是批量数据拷贝和批量数据初始化操作,因为这两类批量操作需要将大量数据在内存通道中来回传输,造成系统延迟高、能效低。新型的数据库应用、

加密算法以及图像处理等方面,需要执行大量的按位操作,加速这类操作能够从根本上降低这些应用的操作延迟。无论是系统级,还是新型应用级,优化这些批量的关键操作,都能够降低系统的延迟,提升系统能效,但是实现的过程中需要综合考虑降低硬件开销、简化编程人员的开发难度等方面的因素。针对这些系统级的关键操作和新型的应用场景,设计优化关键操作的 DRAM 架构会是未来的研究方向。

表 1　低延迟 DRAM 架构的汇总

低延迟类型	架构名称	特点	额外硬件开销	新指(命)令	容量变化	适用场景
优化关键操作	RowClone	Subarray 内数据直接传输	控制逻辑	memcopy meminit	不变	拷贝和初始化操作密集的应用
	LISA	相邻 Subarray 间直接传输数据	隔离晶体管	RBM	不变	批量数据拷贝频繁
	Ambit	芯片内部执行按位操作	修改读出放大器	无	不变	批量按位操作
降低平均访存延迟	CHARM	频繁访问的数据位于快速 Subarray 的时间更长	重组列解码器和 Bank 间的数据线	无	不变	操作系统协助管理应用程序的阶段变化
	DAS-DRAM		快/慢 Subarray 的位线长度和各自所占比例	无	inclusive-cache 减少 1/8 容量	内存需求适中
	TL-DRAM	位于近段数据的访问延迟低	隔离晶体管	无	inclusive-cache 减少 1/4 容量	内存需求适中
	Son 等人	结合两种读出放大器管理策略的优势		无	不变	访存密集
	MCR	利用高电荷量的优点	修改外围电路逻辑	修改 MRS	减少 1/2 ~ 3/4	内存需求适中
提升请求并发度	SALP-2	请求操作能够部分重叠	每个 Subarray 增加一个锁存器,但 MASA 要比 SALP-2 多增加 1 位锁存器	无	不变	通用场景
	MASA	可同时激活两个 Subarray		SA _ SEL	不变	通用场景
	SARP	Subarray 间刷新和访存操作可并发执行	每个 Subarray 增加两个复用器和一个控制块	REFpb	不变	通用场景
	CREAM		Subarray 增加一个复用器,Bank 增加两个寄存器	无	不变	通用场景

由于位线的长度直接决定了单次访存的延迟,许多研究利用位线长度不同的阵列组成访存延迟不同的 Subarray。这些架构需要考虑的因素是让尽可能多的访存请求在低访存延迟的阵列中完成,或者实现低开销的数据迁移机制将位于高访存延迟阵列的数据迁移到低访存延迟阵列。这类方案需要权衡多方面的设计开销,首先是高延迟阵列和低延迟阵列的大小和所占比例,然后需要考虑地址的映射方案,最后还需要增加适当的迁移机制,虽然能够在一定程度上提升系统能效,但是需要整套复杂的设计方案。Son 等人[58]虽然结合读出放大器的管理策略,能够有效降低访存密集的应用场景,但是访存稀疏的情况下会增加请求访存延迟。MCR 更是以牺牲大量的内存容量为代价,利用 DRAM 单元高电荷量的优点实现低延迟的访存过程,这是不适用于数据中心中对内存需求量大的应用场景。

正如章节 1.1 所介绍,当前 DRAM 系统的完全并行仅限于通道级别,而 Rank 级和 Bank 级属于部分并行,且 Bank 级是可并行的最小级。对于相同 Bank 内部的连续请求,只能串行处理,所以提升 Bank 内或者 Subarray 间请求的并发性能够有效地提升系统能效。SALP-2 和 MASA 开发 Subarray 间请求可重叠的部分,增加相同 Bank 内读写请求的重叠时间。另外,SARP 和 CREAM 利用 Subarray 内部的读出放大器实现 Subarray 级刷新,实现 Subarray 间刷新操作和读写操作的并行。上述提升请求并发性的架构能够有效地降低访存延迟,基于已有的这些架构,如何区分应用程序的类型,进一步降低延迟敏感型应用的延迟和能耗会是未来值得研究的方向。

4 低功耗的 DRAM 架构

重新组织或者修改 DRAM 的内部形式的低功耗 DRAM 架构,主要是通过更细粒度地访问和管理实现系统功耗的降低。

4.1 细粒度的激活方案

激活功耗主要依赖于被激活的 MAT 数,激活的 MAT 数越少,引入的激活功耗也就越少,如图 5 所示。由于 Subarray 中 MAT 间共享的行激活总线和行解码器等需要一定的能量开销,所以随着所激活的 MAT 数减少,激活功耗也不会按比例降低。

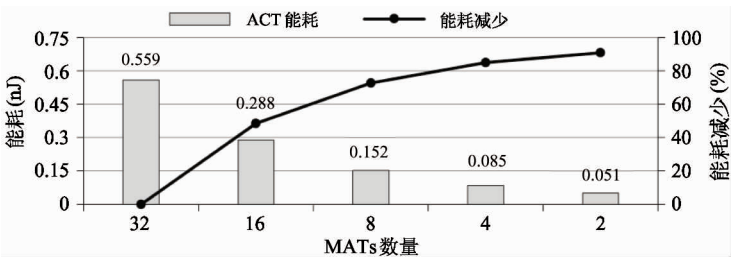

图 5 行激活功耗与被激活的 MAT 数的关系[10]

FCRAM(fast cycle RAM)[63]是通过将 DRAM 阵列划分成更小的子阵列,实现细粒度 DRAM 行访问的最早研究。FCRAM 基于富士通公司独家的设计规范,将 DRAM 阵列分割成多个更小的子阵列,当 DRAM 芯片收到一条激活命令后,将不再激活原来的整个 DRAM 行,而是只激活子阵列,并将子阵列中的数据发送到相应的读出放大器。这样就可以消耗更少的能量和花费更短的时间完成激活操作。

Fine-grained activation[64]通过添加额外的控制逻辑,利用 posted-CAS 命令,在激活 DRAM 阵列时与 FCRAM 类似,不需要激活整行,而是只选择激活该行的部分(后续会将行的部分称作“子行”,不过不同的研究方案中,子行的大小也不相同)。由于 posted-CAS 命令是在行地址发出一个周期后紧接着发出列地址,这样在整行被激活之前,列地址信号就已经到达。为了使用 posted-CAS 命令后,DRAM 设备上增加额外的逻辑用于缓存列地址,列地址紧挨着激活操作后基本上就可以使用,这样 DRAM 芯片就只需要激活所选中的 DRAM 行中真正需要被访问的部分,从而减少激活功耗。另外,该方案能够在两个相邻的时钟周期内完成激活、列访问以及预充电等 3 个操作,简化了内存控制器的命令总线调度。

Mini-Rank[65]的设计思路是为每个 DIMM 增加一个 MRB(mini-Rank buffer)桥接芯片,用于将 x64 Rank(为了组成一个 64 位数据通路的 DDRx,可以使用 4 个 x16 的芯片组成一个 x64 的 Rank)分解成更小的 Rank,比如 x32、x16 以及 x8. 对于 x16 的芯片,每个 x32 的 mini-Rank 包含 2 个芯片,同理,x16 的 mini-Rank 则只包含一个芯片。MRB 直接连接 DDRx 总线,可以直接接收内存命令和大多数的控制信号。MRB 与每个 DRAM 芯片之间都有单独的数据链接。MRB 的主要功能是实现 DDRx 总线与 DRAM 芯片间的可靠数据传输,并为芯片生成片选信号。Mini-Rank 能够减少每次激活操作需要激活的芯片个数,所以可以有效地节约激活功耗。此外,Mini-Rank 的数量相比于原来的 Rank 数呈倍数的增加,也就增加了设备更快进入低功耗模式的可能性,从而降低背景功耗。

SSA(single Subarray access)[66]重新组织 DRAM 内部阵列的布局,为每个 Subarray 增加外围电路,并重新设计数据与这些阵列之间的映射关系,使得内存控制器可以从单个 DRAM 芯片中获取所需的整个缓存行(cache line)数据。该方案需要修改现有内存控制器的接口,将通道划分为多个带宽较低的通道。现有 DRAM 阵列支持的低功耗模式的粒度很粗,通常都是以整个 Rank 为粒度的,在处理访存较密集的数据集时,基本上就不会进入这种低功耗的状态。SSA 以较细粒度的芯片为单位进行资源访问,从而能够提供更多的可能性让芯片处于低功耗状态。

MSRB (multiple sub-row buffers)[67] 结构将 DRAM Bank 中原有的单个大的行缓存切分成多个小行缓存,每个小行缓存大小一致,并实现子行粒度的 DRAM 行激活等操作。内存控制器需要记录每个 Bank 中当前可用的小行缓存,当一个新的子行被激活以后,将子行中的数据读取到其中一个小行缓存,然后将该子行进行预充电操作,完成对当前子行的一次访问。由于每次操作的都是子行,所以能够减少激活功耗。另外,MSRB 能够提供缓存更多的数据行,随着子行命中率的提升,可以减少更多的激活和预充电操作,从而节约功耗。由于较高的行命中率能够减少访存的周期,快速完成运行的任务,节省额外的背景功耗。

Half-DRAM[12] 重新组织 DRAM 阵列,将每个 MAT 分成对等的两半,每次激活操作只激活其中的一半。另外,Half-DRAM 执行读写操作时,能够完全利用每个 MAT 中的 HFF,从而可以确保原有的 n 位预取机制。由于 Half-DRAM 每次只操作操作激活的半行,所以能够缓解现有 DRAM 的功耗限制,并保持原有 DRAM 的全带宽。由于半行激活的方案放松了功耗限制条件,通过集成子阵列级别的并行性(SALP),可以实现对同一个 Bank 中位于不同 Subarray 的不同部分的两个半行访问请求(一个请求访问的行在左边一半,另一个请求访问在右边的一半)并发地发出,从而提升整个内存系统性能。

MCDIMM(multicore DIMM)[68] 将 DRAM 的芯片重新组织成多个虚拟内存芯片,每个虚拟内存芯片有单独的数据通路,芯片间共享一条命令通路,用于接收不同的地址和控制信号。内存控制器为所选择的一组芯片发出独立的片选信号。MCDIMM 将原有存储该片选信号的寄存器替换为 Demux 寄存器。Demux 寄存器收到命令信号后,会将该信号路由到合适的 DRAM 芯片,而不是像原来那样简单地重复或者广播该命令信号。因此,每个芯片通过共享的命令总线能够收到不同的命令,不同的芯片可以用私有的数据总线并发地传输数据。每个 DRAM 通过一个单独的窄数据总线和内存控制器通信,该总线也被其他 MCDIMMs 中相应的芯片所共享。MCDIMMs 每次访存操作只激活一个 DRAM 芯片,从而节省激活功耗。与传统的 DRAM 结构相比,拥有更小的页面和更多的 Bank。当访存序列随机的情况下,Bank 数越多,造成访问 Bank 冲突的情形也就越少,从而也有益于性能的提升。

4.2 低频率与低功耗的芯片架构

Decoupled DIMM[69] 从总线上解耦合内存控制器和 DRAM 芯片,让内存总线的数据传输速率远高于 DRAM 芯片的传输速率,从而使用功耗和数据速率都相对较低的 DRAM 芯片构建高带宽的内存系统。实现 Decoupled DIMM 架构的前提是内存通道包含多个 DIMM 或者每个 DIMM 具有多个 Rank,因为只有在这样的系统中,所有芯片的内存带宽是总线带宽的 2 倍或 2 倍以上(即组合多个数据速率低的 DRAM 芯片的带宽可以匹配内存总线的带宽)。Decoupled DIMM 增加一个被称作“同步缓冲区”的芯片,用于在内存总线和 DRAM 芯片之间中转数据,以确保 DRAM 芯片以更低的数据速率运行。DRAM 芯片的数据传输速率是可配置的,设备默认情况下以总线数据速率的一半运行。此外,内存控制器需要修改访存的调度方式,以避免可能由数据速率的不同而引入的潜在访问冲突。

BOOM(buffered output on module)[70] 利用多个低功耗、低频率的移动存储芯片组成一个可以满足服务器对高带宽、高可靠性的 DIMMs 的需求。每个 DIMM 中新增一个简单的缓冲芯片,用于桥接内部慢速高带宽数据总线和外部的快速低带宽数据总线。该缓冲芯片还包含一组同步队列,用于处理外部总线和内部总线之间突发长度的差异。由于使用了频率更低的移动端芯片,所以 DRAM 系统的背景功耗可以降低。BOOM 使用很宽的内部数据通路,以避免连续访存请求在 Rank 间切换的开销,可以进一步减少功耗和执行时间。

Malladi 等人[9] 分析数据中心应用的资源利用率发现,许多服务器的内存带宽没有得到充分的利用。基于该发现,他们利用低功耗的 LPDDR2 移动设备芯片替换数据中心 DDR3 内存,以降低系统的峰值带宽为代价,降低内存系统的背景功耗。为了保持通道的数量和容量不变,需要增加芯片的容量或者每个通道的芯片数量。前者会增加芯片封装的

难度,所以通过增加每个通道的芯片数量,但是简单地增加芯片会导致通道过载,并影响信号的完整性。为了确保信号的完整性,在通道和 DRAM 芯片之间增加一个能够缓存和重新计时 DQ 和 CA 总线的缓存,该缓存被称为“Buffered LPDDR2 DIMMs”。芯片和缓存间增加点对点的链路,这样无论容量是否增加,都能够保证可靠通信,同时减少了通道的负载。该方案是以增加延迟和引脚为代价,让通道的容量翻倍。

4.3 优化写操作功耗

SDS (skinflint DRAM system)[71] 修改现有 DRAM 系统的组织方式,为每个芯片提供单独的片选信号和时钟使能信号,并添加额外的缓存,实现芯片级 DRAM 写访问。与现有 DRAM 系统相比,SDS 不需要同时访问 Rank 中的所有芯片,只需要访问被选中的部分芯片,因为 SDS 支持单独地访问和控制每个 DRAM 芯片。SDS 为每个缓存行增加一个被称作“片写向量”(chip write vector) 的硬件结构,用于指示写回每个芯片的数据是否真正地被修改。当 SDS 在芯片间执行激活 DRAM 行操作时,首先比较缓存行中当前新数据和先前的数据,并将比较结果存储到 CWV。如果 CWV 指示返回当前芯片的所有数据都未被修改时,则该芯片不会被访问并继续处于不活动状态,从而减少激活/预充电和写入功耗以及空闲功耗。

PRA (partial row activation) 架构[10] 是一种优化内存写入的细粒度行激活方案,通过重新设计 DRAM 的组织形式,缓解 DRAM 行的读取过度的问题,同时减少 DRAM 系统的激活功耗。PRA 区别处理读操作和写操作的行激活过程:对于读取请求,激活整行并保持 n 位预取能力来克服细粒度激活的带宽下降;对于写请求,不需要将整行数据写回,而是将被修改的 cache line 数据写回,只激活 DRAM 行中被修改的部分(以 MAT 为单位),从而降低了内存写入的 I / O 功耗。由于写请求具有很差的读出放大器局部性,所以很少出现连续的写请求访问相同的被激活的 DRAM 行。对于写请求,被写回的 64 字节 cache line 包含的所有字并不是都被修改过的,未被修改过的字所在行的部分就不需要激活,只需要将修改过的字写回即可,从而减少 I/O 功耗。为了支持 PRA 架构,需要 cache 提供字粒度的“脏”(修改)信息,并在 DRAM 芯片上增加一些控制逻辑。

4.4 减少冗余刷新操作

由于制造工艺的差异,DRAM 单元的维持时间不同,ProactiveDRAM[72] 实现 DRAM 自动检测当前芯片中需要执行额外刷新的弱行,并协调内存控制器发出辅助行激活命令。ProactiveDRAM 设计了一个从 DRAM 芯片到内存控制器的后向通信通道。ProactiveDRAM 利用该通道实现混合使用用于正常刷新的自动刷新(auto refresh, AR) 和用于额外刷新补偿的行地址选通刷新(RAS-only refresh, ROR)。ProactiveDRAM 利用布隆过滤器将弱行的信息保存到 DRAM 硬件中,当 DRAM 发现某些弱行需要被执行额外的刷新以补偿其较短的保留时间时,会向内存控制器发送激活命令,然后这些激活命令被插入到内存控制器端的正常命令队列,以便内存控制器执行调度。

4.5 多粒度访存模式

许多数据结构存在多种访问模式,但是物理地址空间的数据布局会出现跨步访问模式(比如按列访问按行存储的矩阵数据),从而浪费内存带宽,并引入很高的访问延迟。AGMS(adaptive granularity memory system)[73] 权衡存储效率、细粒度的吞吐量以及能效这 3 个方面,将内存访问划分为粗粒度和细粒度 2 种模式,对于空间局部性高的存储区域使用粗粒度配置,而空间局部性低的存储区域使用细粒度配置。细粒度的 DRAM 访问模式能够最大限度地减少了片外流量,并且可以避免传输不必要的数据来降低 DRAM 功耗,同时提高了 DRAM 访问的并发性和整体性能; 粗粒度的 DRAM 访问模式能够有效地降低缺失率,同时最大限度地减少了控制开销和冗余开销。AGMS 需要多级系统协作完成。首先,需要编程人员为应用程序提供了首选的粒度信息;其次,通过为操作系统增加虚拟内存接口来管理每个页面的访问粒度;然后,利用扇区缓存[74] 管理缓存层次结构中的细粒度数据;最后,利用细粒度 Rank[65] 的内存系统和混合粒度调度策略在 DRAM 系统内部提供有效的多粒度访问。

DGMS(dynamic granularity memory system)[75]与 AGMS 类似,同样以粗、细两种不同的粒度访问内存。但是 DGMS 不依赖于软件或者操作系统,它利用硬件动态预测内存访问的粒度,并动态地改变内存访问模式。DGMS 为每个处理器核心增加一个空间模式预测器和一个局部预测控制器,并在内存控制器中增加一个用于自适应地调整局部预测结果的全局预测控制器。DGMS 是一个纯硬件的机制,不需要预先了解应用程序的特征,也不需要操作系统或者编程人员的参与。

GS-DRAM(gather-scatter DRAM)[76]修改内存控制器,使其能够使用单个读/写命令就能够从不同的芯片访问属于跨步模式的多个值。对于需要访问同一个芯片中的多条数据的访问模式,现有的内存控制器需要对每一条数据进行一次读/写操作,而 GS-DRAM 通过引入数据缓存机制,降低出现这种访问的情形。对于现有 DRAM 的接口,一个读/写命令会访问同一个 Rank 内相同地址的所有芯片。GS-DRAM 不再向每个芯片发送独立的地址,而是为每个跨步模式映射一个模式编号,基于该模式编号,每个 DRAM 芯片使用一个简单的列转换逻辑解析新的列地址,并访问相应的数据片段,从而实现返回的 cache line 包含来自不同芯片的跨度模式的多个值。

4.6 小 结

本小节所述所有的架构都是面向低功耗的研究,表 2 总结了本部分介绍的所有架构。

细粒度的激活方案能够减少每次激活操作所激活的 DRAM 单元数,所以能够最直接地降低激活功耗。由于早期的细粒度激活方案更加偏向于降低激活细粒度的激活方案能够减少每次激活操作所激活的 DRAM 单元数,所以能够最直接地降低激活功耗。由于早期的细粒度激活方案更加偏向于降低激活功耗,而造成较大的硬件开销和总线带宽下降。相较于这些方案,Half-DRAM 降低的单次激活功耗最少,但是能够维持原有的总线带宽。大部分细粒度激活的架构更适用于访存稀疏和访存局部性比较差的应用场景,同时也需要设计灵活的调度算法,将访存请求尽可能地分布到不同的 Bank,减少 Bank 冲突的情形,最大化 Bank 级的并行访问。保障总线带宽的同时,如何以低的硬件开销设计低功耗的细粒度激活的 DRAM 架构仍有很大的研究空间。

表 2 低功耗 DRAM 架构的汇总

低功耗类型	架构名称	芯片类型	额外硬件开销	总线标准	激活阵列大小	适用场景	总线带宽
细粒度激活	FCRAM	DDRx	修改总线和操作协议	富士通标准	单个 MAT	访存局部性差	1/16
	Cooper-Balis	DDRx	缓存列地址的额外逻辑	JEDEC	单个 MAT	访存局部性差	1/16
	Mini-Rank	DDRx	MRB 芯片	JEDEC	多个粒度	访存稀疏	1/2 ~ 1/8
	SSA	DDRx	重设映射关系和内存控制器接口	JEDEC	单个芯片中的单个 MAT	访存局部性差	1/64
	MSRB	DDRx	重组读出放大器	JEDEC	单个 MAT	访存局部性好	1/16
	Half-DRAM	DDRx	控制逻辑和字线驱动器	JEDEC	每个 MAT 的一半	访存局部性差	1
	MCDIMM	DDRx	Demux 寄存器	JEDEC	多种粒度	访存局部性差	1/4 ~ 1/8
低功耗与低频率芯片	Decoupled DIMM	DDRx	同步缓存区	JEDEC	所有 MAT	访存密集	1
	BOOM	LPDDRx	缓存芯片	JEDEC	所有 MAT	访存密集	1
	Malladi 等人	LPDDRx	通道和芯片间缓存	JEDEC	所有 MAT	访存密集	1
优化写操作	SDS	DDRx	片写向量	JEDEC	芯片内的 MAT	写操作频繁	1
	PRA	DDRx	控制逻辑	JEDEC	1/8 ~ 1 MAT	写操作频繁	1
优化刷新操作	ProactiveDRAM	DDRx	反向通信通道	JEDEC	所有 MAT	超大容量内存	1
多粒度访存	AGMS	DDRx	扇区缓存	JEDEC	多种粒度	依赖编程人员	1
	DGMS	DDRx	扇区缓存、预测和控制器	JEDEC	多种粒度	访存局部性差	1
	GS-DRAM	DDRx	列转换逻辑、控制逻辑	JEDEC	所有 MAT	访存密集	1

低频率和低功耗的芯片能够从根本上减少系统的背景功耗,但是为了匹配内存控制器和 DRAM 芯片的速率,需要增加额外的硬件同步缓冲区(缓冲芯片),从而会增加存取延迟和功耗。另外,随着 DDRx 的发展,新的低功耗特征不断出现,采用 LPDDRx 芯片实现低功耗的方案将不再有太多的优势。

由于写操作具有很差的读出放大器局部性,并且写回的数据并不是全部被修改过,所以 SDS 和 PRA 对于写操作以不同的粒度写回被修改过的部分。ProactiveDRAM 通过增加从 DRAM 芯片到内存控制器的后向通信通道,实现对 DRAM 单元差异化的刷新机制,从而降低刷新带来的功耗。当前对于写操作和刷新操作的优化相对比较独立,所以基于 DRAM 单元的特性和写操作的执行过程,如何优化写操作和刷新操作的功耗也是未来值得进一步研究的问题。

为了提升跨步访问物理地址空间数据的性能,AGMS 和 DGMS 架构分别以静态和动态的方式实现多粒度的访存模式,不但需要细粒度访存接口和扇区缓存的支持,同时会造成很大的内存接口和 Cache 标签存储开销。GS-DRAM 降低了硬件开销,只需要简单修改现有 Cache,但是需要 Cache、指令集以及软件的协同支持,或者处理器动态识别应用的访问模式。虽然这些方案能够有效地提升跨步访问模式的性能,但是它们局限于应用场景,并且要么需要扇区缓存的支持,要么需要多层系统的支持,要么需要硬件动态识别程序访问特征,因此如何降低硬件开销、简化软硬件协同设计,是未来关于跨步访问模式研究亟需解决的问题

5 结 论

具有高带宽、低延迟、高密度、低成本等特性的动态随机存储器(DRAM)已广泛应用于众多计算系统。DRAM 主存系统所占总功耗的比率也在逐渐增加。为了优化 DRAM 内存系统的能效,可以通过修改操作系统、内存控制器以及 DRAM 架构实现。本文首先对当前 DRAM 架构的背景进行介绍;然后从低功耗状态转换方案、请求调度方案、刷新调度方案、内存控制器级刷新减少方案、操作系统级刷新减少方案以及电荷量优化方案等 6 个方面出发,概述了通过修改操作系统和内存控制器实现高能效内存系统的研究;之后将通过修改 DRAM 架构实现高能效内存系统的研究分为低延迟的 DRAM 架构和低功耗的 DRAM 架构两大部分进行介绍。

综合现有不同层次优化内存系统能效的研究,修改 DRAM 架构是提升 DRAM 能效最直接、有效的方法。基于上述第 3 部分和第 4 部分的介绍和分析,得到以下主要结论:(1) 针对系统级的关键操作和新型的应用场景,设计优化关键操作的 DRAM 架构会是未来的研究方向。(2) 基于提升请求并发性的架构,通过区分应用程序的类型,降低延迟敏感型应用的延迟和能耗会是未来值得研究的方向。(3) 保障总线带宽的同时,以低的硬件开销设计低功耗的细粒度激活的 DRAM 架构依旧有很大的研究空间。(4) 基于 DRAM 单元的特性和写操作的执行过程,优化写操作和刷新操作的功耗值得进一步研究。(5) 硬件开销低、软硬件协同过程间的跨步访问模式也是值得深入研究的方向。尽管需要考虑硬件开销、设计复杂度、能效以及编程人员的开发难度等众多因素,但是通过修改 DRAM 架构的研究依旧是未来优化内存系统能效的主要研究方向之一。

参考文献

[1] Chen C L P, Zhang C Y. Data-intensive applications, challenges, techniques and technologies: a survey on big data[J]. *Information Sciences*, 2014, 275: 314-347

[2] Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing[C]. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, Indianapolis, USA, 2010. 135-146

[3] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing[C]. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, San Jose, USA, 2012. 2-2

[4] Chung E S, Milder P A, Hoe J C, et al. Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs? [C]. In: Proceedings of the 43rd Annual IEEE/ACM International Symposium on

Microarchitecture, Atlanta, USA, 2010. 225-236

[5] Joao J A, Suleman M A, Mutlu O, et al. Bottleneck identification and scheduling in multithreaded applications [J]. *ACM SIGPLAN Notices*, 2012, 47(4): 223-234

[6] Mutlu O. Memory scaling: a systems architecture perspective[C]. In: Proceedings of the 5th IEEE International Memory Workshop, Monterey, USA, 2013. 21-25

[7] Lim K, Chang J, Mudge T, et al. Disaggregated memory for expansion and sharing in blade servers[C]. In: Proceedings of the 36th Annual International Symposium on Computer Architecture, New York, USA, 2009. 267-278

[8] Barroso L A, Clidaras J, Hölzle U. The datacenter as a computer: an introduction to the design of warehouse-scale machines[J]. *Synthesis lectures on computer architecture*, 2013, 8(3):1-154

[9] Malladi K T, Lee B C, Nothaft F A, et al. Towards energy-proportional datacenter memory with mobile DRAM [J]. *ACM SIGARCH Computer Architecture News*, 2012, 40(3): 37-48

[10] Lee Y, Kim H, Hong S, et al. Partial row activation for low-power dram system[C]. In: Proceedings of the 23rd IEEE International Symposium on High Performance Computer Architecture, Austin, USA, 2017. 217-228

[11] Ha H, Pedram A, Richardson S, et al. Improving energy efficiency of dram by exploiting half page row access [C]. In: Proceedings of the 49th International Symposium on Microarchitecture, Taipei, China, 2016. 1-12

[12] Zhang T, Chen K, Xu C, et al. Half-DRAM: A high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation[C]. In: Proceedings of the 41st annual international symposium on Computer architecture, Minneapolis, USA, 2014. 349-360

[13] Yoon D H, Chang J, Muralimanohar N, et al. BOOM: Enabling mobile memory based low-power server DIMMs [C]. In: Proceedings of the 39th Annual International Symposium on Computer Architecture, Portland, USA, 2012. 25-36

[14] Jacob B, Ng S, Wang D. Memory Systems: Cache, DRAM, Disk[M]. Burlington: Morgan Kaufmann Publishers, 2010. 315-320

[15] Micron Inc. 2gb: x4, x8, x16 ddr3 sdram features[EB/OL]. https://perso.esiee.fr:Micron, 2009

[16] Lee D, Kim Y, Seshadri V, et al. Tiered-latency DRAM: a low latency and low cost DRAM architecture [C]. In: Proceedings of the 19th International Symposium on High Performance Computer Architecture, Shenzhen, China, 2013. 615-626

[17] Bhati I, Chang M T, Chishti Z, et al. DRAM refresh mechanisms, penalties, and trade-offs[J]. *IEEE Transactions on Computers*, 2016, 65(1): 108-121

[18] Bhati I S. Scalable and energy efficient DRAM refresh techniques[D]. Maryland: College Park University of Maryland, 2014. 20-32

[19] Skhynix. DDR4 SDRAM device operation[EB/OL]. https://www.skhynix.com:Skhynix, 2018

[20] Delaluz V, Kandemir M, Vijaykrishnan N, et al. Energy-oriented compiler optimizations for partitioned memory architectures[C]. In: Proceedings of the 2000 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, San Jose, USA, 2000. 138-147

[21] Delaluz V, Kandemir M, Vijaykrishnan N, et al. DRAM energy management using software and hardware directed power mode control [C]. In: Proceedings of the 7th IEEE International Symposium on High Performance Computer Architecture, Monterrey, Mexico, 2001. 159-169

[22] Fan X, Ellis C, Lebeck A. Memory controller policies for DRAM power management[C]. In: Proceedings of the 2001 International Symposium on Low Power Electronics and Design, Huntington Beach, USA, 2001. 129-134

[23] Wu D, He B, Tang X, et al. RAMZzz: Rank-aware DRAM power management with dynamic migrations and demotions[C]. In: Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, USA, 2012. 1-11

[24] Bi M, Duan R, Gniady C. Delay-hiding energy management mechanisms for dram[C]. In: Proceedings of the 16th International Symposium on High Performance Computer Architecture, Bangalore, India, 2010. 1-10

[25] Hur I, Lin C. A comprehensive approach to DRAM power management[C]. In: Proceedings of the 14th International Symposium on High Performance Computer Architecture, Salt Lake City, USA, 2008. 305-316

[26] Amin A M, Chishti Z A. Rank-aware cache replacement and write buffering to improve DRAM energy efficiency [C]. In: Proceedings of the 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design, Austin, USA, 2010. 383-388

[27] Rixner S, Dally W J, Kapasi U J, et al. Memory access scheduling[J]. *ACM SIGARCH Computer Architecture*

News, 2000, 28(2): 128-138

[28] Mutlu O, Moscibroda T. Stall-time fair memory access scheduling for chip multiprocessors[C]. In: Proceedings of the 40th Annual IEEE/ACM international Symposium on Microarchitecture, Chicago, USA, 2007. 146-160

[29] Mutlu O, Moscibroda T. Parallelism-aware batch scheduling: enhancing both performance and fairness of shared DRAM systems[J]. *ACM SIGARCH Computer Architecture News*, 2008, 36(3): 63-74

[30] Kim Y, Han D, Mutlu O, et al. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers[C]. In: Proceedings of the 16th International Symposium on High Performance Computer Architecture, Bangalore, India, 2010. 1-12

[31] Kim Y, Papamichael M, Mutlu O, et al. Thread cluster memory scheduling: Exploiting differences in memory access behavior[C]. In: Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture, Atlanta, USA, 2010. 65-76

[32] Ipek E, Mutlu O, Martínez J F, et al. Self-optimizing memory controllers: a reinforcement learning approach [J]. *ACM SIGARCH Computer Architecture News*, 2008, 36(3): 39-50

[33] Mukundan J, Martinez J F. MORSE: Multi-objective reconfigurable self-optimizing memory scheduler [C]. In: Proceedings of the 18th International Symposium on High Performance Computer Architecture, New Orleans, USA, 2012. 1-12

[34] Stuecheli J, Kaseridis D, Hunter H C, et al. Elastic refresh: techniques to mitigate refresh penalties in high density memory[C]. In: Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture, Atlanta, USA, 2010. 375-384

[35] Bhati I, Chishti Z, Jacob B. Coordinated refresh: Energy efficient techniques for DRAM refresh scheduling[C]. In: Proceedings of the 2013 International Symposium on Low Power Electronics and Design, Beijing, China, 2013. 205-210

[36] Mukundan J, Hunter H, Kim K, et al. Understanding and mitigating refresh overheads in high-density DDR4 DRAM systems[J]. *ACM SIGARCH Computer Architecture News*, 2013, 41(3): 48-59

[37] Nair P, Chou C C, Qureshi M K. A case for refresh pausing in DRAM memory systems[C]. In: Proceedings of the 19th International Symposium on High Performance Computer Architecture, Shenzhen, China, 2013. 627-638

[38] Song S P. Method and system for selective DRAM refresh to reduce power consumption[P]. US Patent: 6094705, 2000-7-25

[39] Ghosh M, Lee H H S. Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D die-stacked DRAMs[C]. In: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, Chicago, USA, 2007. 134-145

[40] Liu J, Jaiyen B, Veras R, et al. RAIDR: retention-aware intelligent DRAM refresh[J]. *ACM SIGARCH Computer Architecture News*, 2012, 40(3): 1-12

[41] Cui Z, McKee S A, Zha Z, et al. DTail: a flexible approach to DRAM refresh management[C]. In: Proceedings of the 28th ACM International Conference on Supercomputing, Munich, Germany, 2014. 43-52

[42] Bhati I, Chishti Z, Lu S L, et al. Flexible auto-refresh: enabling scalable and energy-efficient DRAM refresh reductions [J]. *ACM SIGARCH Computer Architecture News*, 2015, 43(3): 235-246

[43] Liu J, Jaiyen B, Kim Y, et al. An experimental study of data retention behavior in modern DRAM devices: implications for retention time profiling mechanisms[J]. *ACM SIGARCH Computer Architecture News*, 2013, 41(3): 60-71

[44] Khan S, Lee D, Kim Y, et al. The efficacy of error mitigation techniques for DRAM retention failures: a comparative experimental study[J]. *ACM SIGMETRICS Performance Evaluation Review*, 2014, 42(1): 519-532

[45] Isen C, John L. ESKIMO: E nergy savings using S emantic K nowledge of I nconsequential M emory O ccupancy for DRAM subsystem[C]. In: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, New York, USA, 2009. 337-346

[46] Liu S, Pattabiraman K, Moscibroda T, et al. Flikker: saving DRAM refresh-power through critical data partitioning[J]. *ACM SIGPLAN Notices*, 2012, 47(4): 213-224

[47] Venkatesan R K, Herr S, Rotenberg E. Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM[C]. In: Proceedings of the 12th IEEE International Symposium on High Performance Computer Architecture, Austin, USA, 2006. 155-165

[48] Baek S, Cho S, Melhem R. Refresh now and then[J]. *IEEE Transactions on Computers*, 2014, 63(12): 3114-3126

[49] Kotra J B, Shahidi N, Chishti Z A, et al. Hardware-software co-design to mitigate dram refresh overheads: a case for refresh-aware process scheduling[C]. In: Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems, Xi'an, China, 2017. 723-736

[50] Shin W, Yang J, Choi J, et al. NUAT: a non-uniform access time memory controller[C]. In: Proceedings of the 20th International Symposium on High Performance Computer Architecture, Orlando, USA, 2014. 464-475

[51] Hassan H, Pekhimenko G, Vijaykumar N, et al. Charge cache: reducing DRAM latency by exploiting row access locality[C]. In: Proceedings of the 22nd International Symposium on High Performance Computer Architecture, Barcelona, Spain, 2016. 581-593

[52] Zhang X, Zhang Y, Childers B R, et al. Restore truncation for performance improvement in future DRAM systems[C]. In: Proceedings of the 22nd International Symposium on High Performance Computer Architecture, Barcelona, Spain, 2016. 543-554

[53] Seshadri V, Kim Y, Fallin C, et al. RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization[C]. In: Proceedings of the 46th International Symposium on Microarchitecture, Davis, USA, 2013. 185-197

[54] Chang K K, Nair P J, Lee D, et al. Low-cost inter-linked subarrays (LISA): Enabling fast inter-subarray data movement in DRAM[C]. In: Proceedings of the 22nd International Symposium on High Performance Computer Architecture, Barcelona, Spain, 2016. 568-580

[55] Seshadri V, Lee D, Mullins T, et al. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology[C]. In: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, Massachusetts, USA, 2017. 273-287

[56] Son Y H, Seongil O, Ro Y, et al. Reducing memory access latency with asymmetric DRAM bank organizations [C]. In: Proceedings of the 40th Annual International Symposium on Computer Architecture, New York, USA, 2013. 380-391

[57] Lu S L, Lin Y C, Yang C L. Improving dram latency with dynamic asymmetric subarray[C]. In: Proceedings of the 48th International Symposium on Microarchitecture, Waikiki, USA, 2015. 255-266

[58] Son Y H, Seongil O, Ro Y, et al. Reducing memory access latency with asymmetric DRAM bank organizations [C]. In: Proceedings of the 40th Annual International Symposium on Computer Architecture, New York, USA, 2013. 380-391

[59] Choi J, Shin W, Jang J, et al. Multiple clone row DRAM: a low latency and area optimized DRAM[C]. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, USA, 2015. 223-234

[60] Kim Y, Seshadri V, Lee D, et al. A case for exploiting subarray-level parallelism (SALP) in DRAM[C]. In: Proceedings of the 39th Annual International Symposium on Computer Architecture, Portland, USA, 2012. 368-379

[61] Chang K K W, Lee D, Chishti Z, et al. Improving DRAM performance by parallelizing refreshes with accesses[C]. In: Proceedings of the 20th International Symposium on High Performance Computer Architecture, Orlando, USA, 2014. 356-367

[62] Zhang T, Poremba M, Xu C, et al. CREAM: A concurrent-refresh-aware DRAM memory architecture[C]. In: Proceedings of the 20th International Symposium on High Performance Computer Architecture, Orlando, USA, 2014. 368-379

[63] Sato Y, Suzuki T, Aikawa T, et al. Fast Cycle RAM (FCRAM); a 20-ns random row access, pipe-lined operating DRAM[C]. In: Proceedings of the 1998 Symposium on VLSI Circuits, Honolulu, USA, 1998. 22-25

[64] Cooper-Balis E, Jacob B. Fine-grained activation for power reduction in DRAM[J]. *IEEE Micro*, 2010, 30(3): 34-47

[65] Zheng H, Lin J, Zhang Z, et al. Mini-rank: Adaptive DRAM architecture for improving memory power efficiency[C]. In: Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture, Lake Como, Italy, 2008. 210-221

[66] Udipi A N, Muralimanohar N, Chatterjee N, et al. Rethinking DRAM design and organization for energy-constrained multi-cores[C]. In: Proceedings of the 37th Annual International Symposium on Computer Architecture, Saint-Malo, France, 2010. 175-186

[67] Gulur N D, Manikantan R, Mehendale M, et al. Multiple sub-row buffers in DRAM: unlocking performance and energy improvement opportunities[C]. In: Proceedings of the 26th ACM International Conference on Supercomputing, San Servolo Island, Italy, 2012. 257-266

[68] Ahn J H, Leverich J, Schreiber R, et al. Multicore

DIMM: an energy efficient memory module with independently controlled DRAMs[J]. *IEEE Computer Architecture Letters*, 2009, 8(1): 5-8

[69] Zheng H, Lin J, Zhang Z, et al. Decoupled DIMM: building high-bandwidth memory system from low-speed DRAM devices[C]. In: Proceedings of the 36th Annual International Symposium on Computer Architecture, New York, USA, 2009. 255-266

[70] Yoon D H, Chang J, Muralimanohar N, et al. BOOM: Enabling mobile memory based low-power server DIMMs [C]. In: Proceedings of the 39th Annual International Symposium on Computer Architecture, Portland, USA, 2012. 25-36

[71] Lee Y, Kim S, Hong S, et al. Skinflint DRAM system: Minimizing DRAM chip writes for low power[C]. In: Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture, Shenzhen, China, 2013. 25-34

[72] Wang J, Dong X, Xie Y. Proactive DRAM: a DRAM-initiated retention management scheme[C]. In: Proceedings of the 32nd IEEE International Conference on Computer Design, Seoul, Korea, 2014. 22-27

[73] Yoon D H, Jeong M K, Erez M. Adaptive granularity memory systems: a tradeoff between storage efficiency and throughput[C]. In: Proceedings of the 38th Annual International Symposium on Computer Architecture, San Jose, USA, 2011. 295-306

[74] Liptay J S. Structural aspects of the system/360 model 85, II: the cache[J]. *IBM Systems Journal*, 1968, 7 (1): 15-21

[75] Yoon D H, Jeong M K, Sullivan M, et al. The dynamic granularity memory system[C]. In: Proceedings of the 39th Annual International Symposium on Computer Architecture, Portland, USA, 2012. 548-559

[76] Seshadri V, Mullins T, Boroumand A, et al. Gather-scatter DRAM: in-DRAM address translation to improve the spatial locality of non-unit strided accesses[C]. In: Proceedings of the 48th International Symposium on Microarchitecture, Waikiki, USA, 2015. 267-280

A survey of optimizing the energy efficiency of memory system on DRAM architectures

Zhan Xusheng[*,**], Bao Yungang[*], Sun Ninghui[*]

([*] Institute of Computing Technology, Chinese Academy of Science, Beijing 100190)

([**] University of Chinese Academy of Sciences, Beijing 100049)

Abstract

The research status of optimizing the energy efficiency of memory systems in different levels is introduced, and the mechanisms of modifying DRAM architectures are discussed in detail. We outline the research on energy efficiency of DRAM systems by modifying memory controller and operating systems. We emphatically introduce the research about optimizing the energy efficiency through modifying the DRAM architecture. These studies are divided into two groups: 'low-delay DRAM architecture' and 'low-power DRAM architecture'. Low-latency architectures contains three aspects: optimizing key operations, reducing average access latency, and increasing requests concurrency. Low-power architecture studies include five types: fine-grained activation architecture, low-power and low-frequency chips, optimized write operations, optimized refresh operations, and multi-granularity access mode. We show the summary and prospect of optimizing the memory energy efficiency of DRAM architectures in the end.

Key words: memory, dynamic random access memory (DRAM), memory controller, architecture, energy efficiency, low latency, low power consumption