基于多级磁自旋存储器的Cache调度策略的设计

朱艳娜[1] 王党辉[2]
(中国空空导弹研究院 河南 洛阳 471009)[1] (西北工业大学 西安 710129)[2]

摘 要 多级磁自旋存储器(Multi-Level Cell Spin-Transfer Torque RAM,MLC STT-RAM)可在一个存储单元中存储多个比特位,有望取代SRAM用于构建大容量低功耗的最后一级Cache(Last Level Cache,LLC)。MLC STT-RAM的静态功耗在理论上为0,且拥有高密度和优秀的读操作特性,但它的缺陷在于低效的写操作。针对这一问题,在MLC STT-RAM Cache hard/soft逻辑分区结构的基础上,实现了MLC STT-RAM LLC写操作密集度预测技术以及相应Cache结构的设计。通过动态预测写操作密集度较高的Cache块,帮助MLC STT-RAM LLC减少执行写操作的代价。预测的基本思想是利用访存指令地址与相应Cache块行为特征的联系,根据预测结果决定数据在LLC中的放置位置。实验结果显示,在MLC STT-RAM LLC中应用写操作密集度预测技术,使得写操作动态功耗降低6.3%的同时,系统性能有所提升。

关键词 多级磁自旋存储器,最后一级高速缓存,低功耗,预测机制
中图法分类号 TP332.3 **文献标识码** A

Design of Cache Scheduling Policies Based on MLC STT-RAM

ZHU Yan-na[1] WANG Dang-hui[2]
(China Air-borne Missile Academy,Luoyang,Henan 471009,China)[1]
(Northwestern Polytechnical University,Xi'an 710129,China)[2]

Abstract Multi-level cell (MLC) STT-RAM which can store multiple bits per cell,has been considered as a promising alternative to SRAM for the last-level Cache. MLC STT-RAM can reduce static power consumption significantly and has smaller cell size facilitates and better read performance. However,a major shortcoming of MLC STT-RAM Cache is its inefficient write operations. Based on hard/soft partition structure,this paper implemented write intensity prediction for energy-efficient MLC STT-RAM LLC. The objective of this architecture is to dynamically predict whether blocks will be written more than certain times thereby helping to reduce write latency and energy of MLC STT-RAM Cache. The key idea to solve this problem is to correlate write intensity with memory access instruction addresses. On top of that,this paper designed MLC STT-RAM LLC based on this predictor,in which prediction results are used to determine Cache line placement. Experimental results showed that this architecture reduces 6.3% of write energy consumption and improves system performance by 1.9% on average compared to the previous approach.

Keywords Multi-Level Cell Spin-Transfer Torque-RAM, Last level high-speed cache, Energy-efficient, Prediction mechanism

1 引言

为了满足日益增长的计算需求以及与终端用户之间的交流,计算机处理器的计算能力愈来愈强,这导致存储墙问题不容忽视。Cache作为协调存储器与主存间巨大速度差的重要部件,同样需要得到与处理器计算能力相匹配的提升。理论上,Cache容量的大幅增长有效缓解了存储墙问题,但容量的增长使得Cache的功耗增大。Cache的功耗分为静态功耗和动态功耗两部分,工艺水平的不断发展导致静态功耗变得日益显著,在100nm工艺下,静态功耗已占据总体功耗的大部分,甚至有时静态功耗的大小已经远超动态功耗[1]。

处理器设计者既希望使用大容量的Cache,同时又希望获得能有效解决静态功耗问题的方法,但构建出一个同时拥有高性能和低功耗特点的Cache极具挑战性。传统存储器SRAM对于制造更大容量的Cache并不具有优势,原因在于其密度较低、漏电流较高,且在纳米级制造工艺下的可靠性较低。

目前,很多学者致力于研究Phase-Change Memory (PCM)[2-3],Spin-Transfer Torque RAM (STT-RAM)[4-5]等新兴存储器,并将其分别运用于主存或Cache制造。非易失性存储器Multi-Level Cell STT-RAM (MLC STT-RAM)[6-7]的每单元可存储两个比特,即密度相比STT-RAM又增大一倍,使得MLC STT-RAM对于构建大容量、低功耗的片上Cache具有更大的潜力。

L1 Cache由于是存储系统中最上层即最接近CPU的一层。系统需要L1 Cache的响应速度足够快速,且能够与CPU

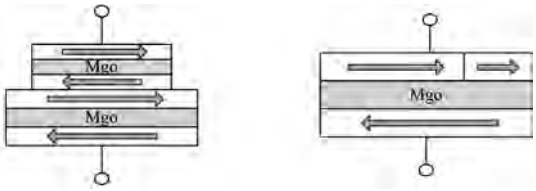
朱艳娜(1992—),女,硕士,助理工程师,主要研究方向为微处理器体系结构与设计,E-mail:zhuyn@mail.nwpu.edu.cn。

的工作速度相匹配。最后一级 Cache，是存储系统中较高级 Cache 与速度最慢的主存之间的衔接部分，因此它的容量大小至关重要，并且会对系统性能产生较大的影响。基于此，研究者们提出在保留 SRAM L1 Cache 的情况下，使用非易失性存储器(Non-Volatile Memory，NVM)取代传统 SRAM 用于 LLC 的制造工作。

2 MLC STT-RAM 存储技术及逻辑分区结构

2.1 MLC STT-RAM 的结构及读写操作

多级磁自旋存储器是一种非易失性存储器。如图 1 所示，MLC STT-RAM 存在两种不同的结构，不同之处在于存储单元中两个 MTJ 的组合位置，分为水平型与垂直型。水平型 MLC STT-RAM 的 hard 位读操作速度较快但写操作速度较慢；soft 位则正好与之相反，写操作速度较快而读操作速度较慢；垂直型 MLC STT-RAM 的 hard 位读、写操作速度相比 soft 位均较慢。本文针对水平型 MLC STT-RAM 进行研究。对于两比特数据而言，最低有效位(The Least Significant Bit，LSB)认为是 soft-bit，最高有效位(The Most Significant Bit，MSB)认为是 hard-bit。



(a)垂直型 MLC STT-RAM 单元 (b)水平型 MLC STT-RAM 单元

图 1 MLC STT-RAM 存储单元结构

2.2 SLC/MLC STT-RAM 与 SRAM 特性的比较

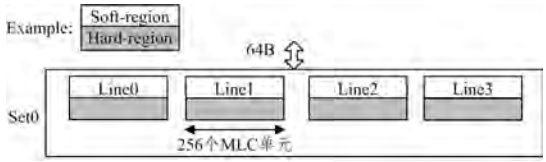
本文使用 CACTI[8] 在已有结论[6,9-10] 的基础上，对 MLC STT-RAM 以及 SRAM 的动态功耗与静态功耗进行建模。当规定 SRAM L2 Cache(即 LLC)的面积开销为 5.1 mm^2 时，其容量为 1MB。MLC STT-RAM 与 SRAM 各项参数的比较如表 1 所列。

表 1 MLC STT-RAM 与 SRAM 特性的比较

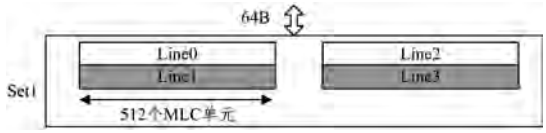
分类	延迟(周期数)	动态功耗(64B)/nJ	静态功耗/W
SRAM(1MB)	查找 tag:1 命中:3	0.31	1.354
MLC STT-RAM (8MB)	查找 tag:3 读命中:5 写命中:37	读:0.32 写:1.58	0

2.3 hard/soft 逻辑分区的详细结构

以四路组相联为例对 hard/soft 逻辑分区[11] 的详细结构进行说明。MLC STT-RAM LLC 中某一 cache set 的原始状态如图 2(a)所示，一个容量为 64B 的物理块包含 256 个 MLC STT-RAM 存储单元，其中 hard-bit 和 soft-bit 各占一半。对两个物理 Cache 块(图中为 Line0 与 Line 1、Line2 与 Line3)进行耦合，并将两个 Cache 块中的所有 MLC STT-RAM 存储单元进行拆分并重组，物理上每个 Cache 块含有一半的 hard-bit 和一半的 soft-bit，但重组后一个逻辑上的 Cache 块如图 2(b)所示，仅含有 hard-bit 或 soft-bit 两者中的一种，不会出现混合的情况。一个 set 中含有一半的 hard 块以及一半的 soft 块。如图 3 所示，每个 Cache 块需要在 tag 部分额外增加 flag 字段，flag 作为标志位且仅含 1 比特，在查找 tag 时使用 0 与 1 区分该 Cache 块是 hard 块还是 soft 块。



(a)MLC STT-RAM cache set 的原始结构



(b)MLC STT-RAM cache set 的分区结构

图 2 MLC STT-RAM hard/soft 逻辑分区

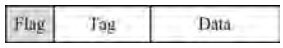


图 3 Cache 块的内部结构

在 MLC STT-RAM LLC 中实现 hard/soft 逻辑分区结构后，沿用表 1 中的面积与容量，MLC STT-RAM LLC 读写操作延迟变化如表 2 所列。虽然所有 hard 块和 soft 块在逻辑上是不相干的，但在对其中的 hard 块执行写操作时，物理上与之相耦合的 soft 块将遭到破坏。soft 块的写操作仅为 19 个周期，大约是原始 MLC STT-RAM LLC 的 50%，但写操作如果发生在 hard 块中，由于破坏耦合 soft 块会带来冗余操作，导致写操作延迟上升至 42 个周期(最后一步对 soft 块的恢复动作不在关键路径上，因此延迟为 5+37 即 42 个周期)，相较于原始 MLC STT-RAM LLC，慢了 14%。

表 2 实施逻辑分区后的读写特性

类型	延迟(周期数)	动态功耗(64B)/nJ	静态功耗/W
MLC STT-RAM (8MB)	查找 tag:3 读命中:5 写命中:37	读:0.32 写:1.58	0
MLC STT-RAM Hard/soft 逻辑分区(8MB)	H-读命中:3 S-读命中:5 H-写命中:42 S-写命中:19	H-读:0.34 S-读:0.38 H-写:1.93 S-写:1.28	0

2.4 动态迁移策略分析

Jiang 等[11] 为了在 hard/soft 逻辑分区结构的基础上进一步提升性能，提出了动态迁移策略。通过使用动态迁移技术进一步使读操作频繁的数据被放置在 hard 块而写操作频繁的数据被放置在 soft 块。动态迁移技术的原理十分简单：如果一块写操作频繁的数据初始时被放在了 hard 块中，则将它迁移到同一 Cache 组的 soft 区域中，选择 soft 区域的 LRU 块并将其中的数据与 hard 块中的数据进行交换；相反，如果一块读操作频繁的数据初始时被放在了 soft 块中，则需要将它迁移到 hard 块中，同时与对方区域的 LRU 块进行数据交换。然而，上述迁移每发生一次，需要对特定 soft 块和特定 hard 块共进行两次读操作和两次写操作；并且在迁移实施过程中，LLC 处于阻塞状态，不能接收和处理新的请求，影响访存的正常进行且减小了有效的访存带宽，对系统性能造成了较大损失。若上述迁移发生的次数过多，则意味着进行写操作的次数会增加，根据前文对 MLC STT-RAM 固有属性的分析可知，动态功耗也会增加。同时，如果存在某些数据既属于读操作频繁型又属于写操作频繁型，显然会导致同一数据在

hard 和 soft 区域之间发生颠簸。

那么如果使用动态预测而不使用动态迁移，在预测准确的情况下即可使得具有相应行为特征的数据被载入到相应区域，同时不会造成额外的动态功耗开销。根据表 1 中的具体数据可知，hard-bit 的读操作延迟较小，但 soft-bit 的读代价与 hard-bit 十分近似，因此本文主要研究针对 MLC STT-RAM LLC 的写操作。若能够在数据因缺失被填充进 Cache 时，通过相关历史信息对该数据将来是否有可能发生较为频繁的写操作实现准确预测，那么可将写操作频繁的数据直接载入到 soft 块中，并且在该数据的整个生命周期内不再对它进行任何迁移操作。通过实施这样的预测，即可利用 MLC STT-RAM 自身的特性来缓解写操作的高延迟和高功耗问题，同时也避免了引入对 LLC 额外的读写操作。

由于指令地址（指令 PC）与程序的行为特征有着密切的联系[12]，因此如果在上述 hard/soft 逻辑分区的基础上通过 PC 信息对 LLC 中数据未来可能的行为特征进行预测，同时若能识别出写操作密集度较高的数据并根据 MLC STT-RAM 存储技术的特性对数据进行相应调度，则可使得 MLC STT-RAM LLC 的写操作代价有所下降。根据 hard/soft 逻辑分区的组织结构，LLC 被分成了两个读写特性不同的区域，且两个区域的容量各占 50%，即读快写慢区（所有的 hard 块）和写快读慢区（所有的 soft 块）。如果能够成功地预测出将来可能承担更多的写操作的数据并将其放置在 soft 区域，即可显著提升 MLC STT-RAM LLC 的写操作效率，同时又能够符合低功耗设计的要求。因此，本文在 MLC STT-RAM LLC 中使用指令 PC 相关的写操作密集度预测机制，将鉴别出的写操作密集度较高的数据放置在 soft 区域，以达到缓解 MLC STT-RAM 写代价的目的。

3　MLC STT-RAM LLC 写操作密集度预测

写操作密集度预测的目标是判定出一个 Cache 块是否属于写操作密集较高的块。用户可以自行设定一个阈值 $W_{threshold}$，如果一个 Cache 块在它的生命周期内执行写操作的次数大于阈值，则认为这是一个写操作密集度高的 Cache 块。在所有触发指令中，如果某些指令更容易引入上述写操作较为频繁的数据使其进入 Cache，则称这样的触发指令为高触发指令；低触发指令代表相反的含义。根据这一规律，可以将鉴别写操作密集度高的 Cache 块转变为鉴别相应的高触发指令。

本文设计的 MLC STT-RAM LLC 写密集度预测机制由两部分组成：追踪触发指令的部分和鉴别高触发指令的部分。

3.1　触发指令的追踪

某个 Cache 块的触发指令是导致 Cache 发生缺失并使该数据被填充进 Cache 的那一条 load 或 store 指令，也是该 Cache 块生命周期内第一条对它进行访问的指令。由上述分析可知，这条触发指令与该块数据的行为特征关系密切。因此，可以通过追踪某一块的触发指令的相关信息来探索和预测该 Cache 块未来可能存在的行为特征。

为了实现这一目标，需要为每个 L2 Cache 块增加一个额外的字段，称其为 T 字段。T 字段即用于存储所在 Cache 块相应的触发指令。当 L1 Cache 发生缺失时，造成此次缺失的指令以及对新数据的请求将一起被送往 L2 Cache（即 LLC）。如果这一请求同样导致 L2 Cache 发生了缺失，那么 L2 Cache 就记录下上述指令信息。因此，使用这种方式能够记录下每个 L2 Cache 块的触发信息，并在生命周期内一直对它进行追踪和监测，直到相应的 Cache 块被淘汰。

图 4 以一系列对 LLC 的访问为例显示了如何对 T 字段的内容进行更新。图 4 中，按照访问 Cache 块的 load/store 指令的先后顺序以及数据的状态变化，由图 4(a)－图 4(h)依次说明了相应 Cache 块的 V（Valid，有效位）字段和 T（Trigger Instruction，触发指令）字段内容的变化情况。图(a)：初始时，设定所有 Cache 块的有效位为 0；图(b)：执行了一条地址为 A 的 store 指令（ST @ A）后，相应 L2 Cache 块的 T 字段立即更新为 A 且有效位置为 1；图(c)和图(d)：在该块的有效位置 1 后，后续任何访问此块的 load 或 store 指令（LD @ B，ST @ C）不再记录信息地址，即 T 字段不发生改变；图(e)和图(f)：即便 L1 Cache 中相应块被淘汰并且有新数据填充进来，只要该块数据仍然驻留在 L2 Cache 中，那么 L2 Cache 块中的 T 字段仍保持不变；图(g)：当 L2 Cache 中的块被淘汰时，V 字段与 T 字段同时清零；图(h)：导致新数据填充进该 Cache 块的 load/store 指令（LD @ D）的地址再次被记录在 T 字段内，作为该新数据的触发指令信息。

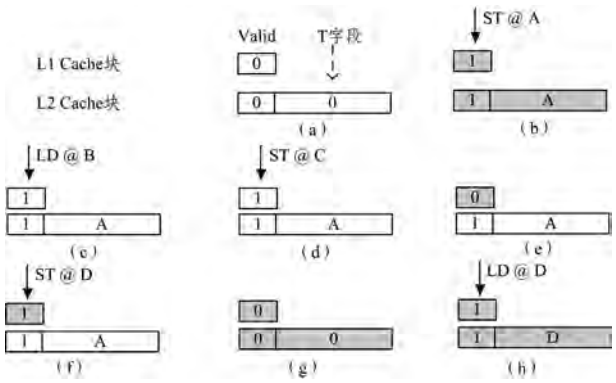


图 4　触发指令追踪及 T 字段更新的过程

3.2　触发指令的鉴别

在记录每个 L2 Cache 块的信息之后，需要鉴别出其中的高触发指令，进而通过这些高触发指令对相应 Cache 块的行为进行预测。本方案的基本思想是追踪每一个 LLC 块的写操作密集度，并将其与该块的触发指令地址联系起来，在未来即可用于预测被相同地址的 load/store 指令载入 LLC 的数据的写操作密集度。

首先，需要统计并保存每个 L2 Cache 块执行写操作的次数。因此，为每个 Cache 块增加一个计数器，计数器的大小以能够储存用户设定的阈值 W 为标准。一个 Cache 块每进行一次写操作，它的计数器的数值就增 1，直到计数器达到阈值 W 后数值不再变化；该块被淘汰时，计数器重置清零。这样就可以监测每个 Cache 块上发生的写操作的数量，从而得出 Cache 块的写操作密集程度，接着就可以利用这些信息对高触发指令进行鉴别。

预测技术需要设定一张预测表，表中每个条目含有两部分内容：触发指令的地址（或地址的哈希值）以及该指令相应的状态（state 字段）。基于上述信息，本文依据不同情况采用不同方式更新相应的 *state* 值，通过 *state* 值判定高触发指令的地址。每个条目中的 state 字段是该指令地址的一个饱和计数器，需要注意的是这个饱和计数器不同于前文所描述的每个 L2 Cache 块中统计写操作数量的计数器。*state*＝10/11

代表该指令是一条高触发指令，*state*＝00/01 代表该指令是一条低触发指令。当 L2 Cache 块中的计数器达到饱和(即阈值 W)时，以 T 字段存储的指令地址哈希值为索引在预测表中查询的相应条目，并将 *state* 的值增 1。相反，如果一个 L2 Cache 块在被淘汰时发生的写操作次数仍小于阈值，则进行查表后将相应的 *state* 值减 1。

为具体说明如何对触发指令进行鉴别，图 5 展示了一系列的操作，体现了 L2 Cache 中特定 Cache 块的附加字段以及它在预测表中相应项目的更新情况。正如上述规定，state＝10/11 表示高触发指令，state＝00/01 表示低触发指令。同时，为便于说明，假设已设定阈值 $W=2$，即 L2 Cache 块计数器的饱和值为 2。当一个 Cache 块执行写操作的次数大于等于 2 时，认为其是一个写操作密集的块。图(a)：设置 L2 Cache 块中的写操作数量计数器(以下简称计数器)的初始值为 0，预测表中相应项的 *state* 饱和计数器(以下简称 *state*)的初始值为 01；图(b)：每执行一次写操作，计数器的值增 1；图(c)：如果发生的是读操作，计数器的值保持不变；图(d)：当计数器的值增加至饱和时，以 T 字段存储的指令地址哈希值为索引在预测表中查找的相应条目，并将 *state* 的值增 1；图(e)、图(f)：一旦计数器的值达到饱和，即使继续有写操作发生，计数器也不再进行更新，直到该块被淘汰，预测表中相应项的 state 数值同样不再进行更新；图(g)、图(h)：如果直到块被淘汰，发生的写操作的次数仍小于阈值，则在淘汰发生时查找预测表并对预测表中相应项的 *state* 做减 1 操作。

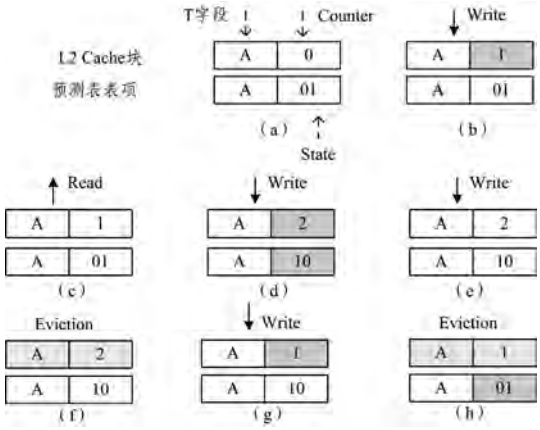


图 5 预测表更新及高触发指令鉴别

3.3 MLC STT-RAM LLC 写操作密集度预测算法

访问 MLC STT-RAM LLC 时，查找 tag 的工作与传统 Cache 中的一致。查找 tag 的工作完成后，存在以下几种可能：

(1)若为读命中，响应请求。从数据域读出数据后送往高一级 Cache，与传统 Cache 的操作完全相同。

(2)若为写命中，首先将数据写进相应 Cache 块的数据域中，同时 Cache 块中的写操作次数计数器增 1。如果此时计数器的值达到了设定的阈值 W，以该块触发指令的地址哈希值作为索引查找预测表，将匹配项的饱和计数器 state 增 1。需要说明的是，对预测表的更新不在关键路径上，因此不会影响 LLC 的响应时间。

(3)若 LLC 发生缺失，首先需要确定新数据的放置位置：

1)如果欲填充的数据为指令，即指令 Cachec(ICache)发生了缺失，将其放置在 hard 块中。

2)如果触发指令为 store 指令，即该块由写缺失引入 LLC，则将其放在 soft 块中。

3)如果触发指令为 load 指令，即该块由读缺失引入 LLC，启动写操作密集度预测算法，以触发指令地址为索引查找预测表，根据表中该指令的 *state* 值决定数据的放置区域。

4)若上述几项条件均不满足，则将数据载入到 hard 块中。

4 实验设置及结果分析

4.1 实验平台和配置

实验在 gem5 体系结构仿真器中通过对原有体系结构的修改与调整，搭建符合 MLC STT-RAM LLC 读写特性的体系结构，并在此基础上实现 MLC STT-RAM LLC hard/soft 逻辑分区结构。在所搭建平台的基础上，集成写操作密集度预测的算法与相关 Cache 结构设计并进行仿真模拟。本文实验配置如表 3 所列，处理器的频率为 2GHz，使用 X86 指令集系统。实验中共包含两级高速缓存，L1 Cache 为 SRAM Cache，指令 Cache 与数据 Cache 分离且均为 32KB，组织结构为 4 路组相联；L2 Cache 为 MLC STT-RAM Cache，16 路组相联；L1/L2 Cache 块的大小为 64B，均使用传统 LRU 替换策略和写回策略。实验中使用 DRAM 构建主存，容量为 4GB。SRAM 与 MLC STT-RAM 的读写延迟与功耗见表 1。

表 3 实验配置

部件	配置信息
CPU core	3GHz，X86
L1 Cache	Split SRAM I/D Cache，32KB，4-way set-associative，64B line size，LRU，write back policy，1-port
L2 Cache	MLC STT-RAM Cache，2 MB，16 way set-associative，64B line size，LRU，write back policy
Main Memory	4GB DRAM，200-cycle latency

4.2 实验结果分析

实验使用 gem5 体系结构仿真器，模拟运行 X86 指令集体系结构编译的 SPEC CPU2006[13] 基准测试程序。仿真过程中，跳过初始阶段并模拟运行 10 亿条指令。

在本文关于 MLC STT-RAM LLC 写操作密集度预测的实验中，Baseline 设置为 Jiang 等提出的在逻辑分区结构上结合动态迁移的调度技术。图 6 显示了本文体系结构与 Baseline 关于动态功耗的对比。

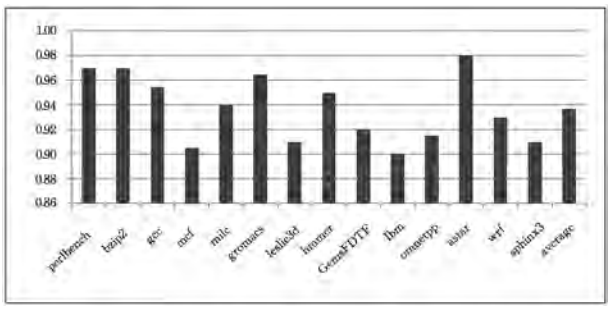


图 6 LLC 写操作功耗对比值

图 6 中以 Baseline 为标准 1，横坐标为实验所使用的不同基准测试程序，纵坐标为 MLC STT-RAM LLC 写操作密集度预测机制与 Baseline 写操作所消耗动态功耗的比值。MLC STT-RAM LLC 写操作密集度预测机制使得 LLC 写操作总功耗的平均值下降了 6.3%，原因在于本文设计的 MLC STT-RAM LLC 写操作密集度预测准确度较高，能够准确地预测出可能承担较多写操作的数据并将其直接放置在 soft 区域中，使得更多的写操作发生在 soft 区域而避免发生在

hard 区域。同时，本文基于 PC 信息对欲填充数据未来的行为特征实现动态预测，且在数据填充进 LLC 时根据预测结果确定放置位置。与 Baseline 相比避免了对数据进行动态迁移造成的功耗开销，在零迁移的基础上确保具有相应读写特征的数据准确放置在 MLC STT-RAM LLC 相应的区域中。

从结果可知，本文体系结构的缺失率与 Baseline 几乎相同。MLC STT-RAM LLC 写操作密集度预测机制中对阈值设置的不同会对 LLC 的缺失率造成一定程度的影响。将阈值设置得越小，就会有越多的 Cache 块被判定为写操作密集度高的块，它们在填充进 LLC 时将被选择放置在 soft 区域中。这样，虽然使得写操作的动态功耗下降得更多，但是由于 MLC STT-RAM LLC 中 soft 区域的容量为 50%，承载能力有限，如果将过多的数据放置在 soft 区域中将导致 LLC 缺失率上升。与之相反，若将阈值设置得过大，则能够避免过多的 Cache 块被选择放入 soft 区域，但可能会漏掉一些写操作较为频繁的数据。

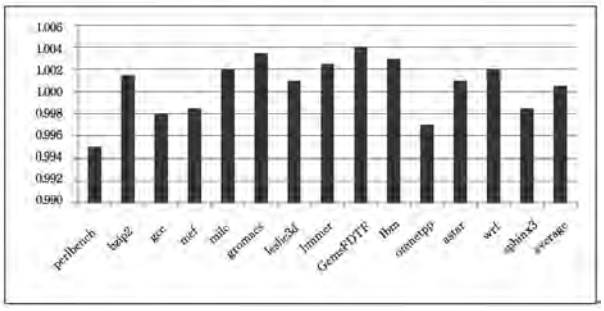


图 7　LLC 缺失率的对比值

实验结果显示，MLC STT-RAM LLC 使用本文写操作密集度预测机制后，系统的总体性能提升了 1.9%，主要由于大量写操作被放置在 soft 区域执行，减少了写操作所需延迟，从而缓解了 Baseline 中写操作延迟较大所造成的 bank 冲突，使得系统的整体性能得到提升。

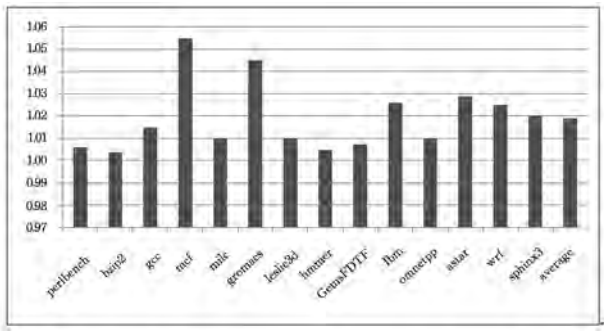


图 8　IPC 对比值

结束语　在 MLC STT-RAM LLC 中，通过使用写操作密集度预测机制，实现了对其的低功耗设计。本文的主旨在于，在 MLC STT-RAM hard/soft 逻辑分区结构的基础上，通过应用写操作密集度预测技术，在数据进入 Cache 前就能较为准确地预测出该数据未来可能进行的写操作密集程度，并以此为依据判定数据放置的区域。预测方案将 Cache 块未来可能的行为特征与其触发指令的地址关联起来，实现了使尽可能多的写操作发生在 soft 块，而尽量减少 hard 块的写操作次数；与此同时，也避免了数据进入 Cache 后在两区域间的动态迁移。实验证明，写操作密集度预测机制所造成的开销可忽略。MLC STT-RAM LLC 使用写操作密集度预测技术后，LLC 中的写操作功耗下降了 6.3%且系统性能提升了 1.9%，对 Cache 缺失率所产生的影响可以忽略。

参 考 文 献

[1] CHANDRAKASAN A P, SHENG S, et al. Low-Power CMOS Digital Design [J]. IEEE Journal of Solid-State Circuits, 1992, 27(4): 473-484.

[2] JOSHI M, ZHANG W Y, LI T. Mercury: A Fast and Energy-Efficient Multi-level Cell based Phase Change Memory System [C]// High Performance Computer Architecture (HPCA). San Antonio: IEEE, 2011: 345-356.

[3] PARK S K, MAENG M K, PARK K W, et al. Adaptive Wear-Leveling Algorithm for PRAM Main Memory with a DRAM Buffer [C]// ACM Transactions on Embedded Computing Systems. New York: ACM, 2014.

[4] EKEN E, ZHANG Y, YAN B. Spin-hall assisted STT-RAM design and discussion [C]// IEEE Magnatics Conference (INTERMAG). Austin: IEEE, 2015: 1.

[5] ZHANG Y J, LI Y, SUN Z Y. Read Performance: The Newest Barrier in Scaled STT-RAM [C]// IEEE Transactions on Very Large Scale Integration (VLSI) Systems. IEEE, 2015: 1170-1174.

[6] CHEN Y R, WONG W F, LI H, et al. Processor Caches with Multi-level Spin-transfer Torque RAM Cells [C]// IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED). Piscataway: IEEE, 2011: 73-78.

[7] BI X Y, MAO M J, WANG D H, et al. Unleashing the potential of MLC STT-RAM Caches [C]// IEEE/ACM International Conference on Computer-Aided Design (ICCAD). San Jose: IEEE, 2013: 429-436.

[8] MURALIMANOLAR N, BALASUBRAMONIAN B, JOUPPI N P, et al. CACTI 6.0: A tool to model large Caches [R]. HP Laboratories, Technical Report, 2009.

[9] CHEN Y R, WANG X B, ZHU W Z. Access Scheme of Multi-Level Cell Spin-Transfer Torque Random Access Memory and its Optimization [C]// IEEE International Midwest Symposium on Circuits and Systems (MWSCAS). Seattle: IEEE, 2010: 1109-1112.

[10] ZHOU P, ZHAO B, YANG J, et al. Energy Reduction for STT-RAM using Early Write Termination [C]// IEEE/ACM International Conference on Computer-Aided Design of Technical Papers. San Jose: IEEE, 2009: 264-268.

[11] JIANG L, ZHAO B, ZHANG Y T, et al. Construction Large and Fast Multi-Level Cell STT-MRAM based Cache for Embedded Processors [C]// Design Automation Conference. New York: ACM, 2012: 907-912.

[12] AHN J W, YOO S J, CHOI K. Write intensity prediction for energy-efficient non-volatile Caches [C]// International Sympo-sium on Low Power Electronics and Design (ISLPED). Beijing: IEEE, 2013: 223-228.

[13] HENNING J L. SPEC CPU2006 benchmark descriptions [J]. ACM SIGARCH Computer Architecture, 2006, 34(4): 1-17.