计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(9):16−23 [doi: 10.15888/j.cnki.csa.005967]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 DRAM 和 PCM 的混合主存模拟器①

张德志, 万寿红, 岳丽华

(中国科学技术大学 计算机科学与技术学院, 合肥 230026)

摘　要: 相变存储器(PCM)由于其非易失性、高读取速度以及低静态功耗等优点, 已成为主存研究领域的热点. 然而, 目前缺乏可用的 PCM 设备, 这使得基于 PCM 的算法研究得不到有效验证. 因此, 本文提出了利用主存模拟器仿真并验证 PCM 算法的思路. 本文首先介绍了现有主存模拟器的特点, 并指出其并不能完全满足当前主存研究的实际需求, 在此基础上提出并构建了一个基于 DRAM 和 PCM 的混合主存模拟器. 与现有模拟器的实验比较结果表明, 本文设计的混合主存模拟器能够有效地模拟 DRAM 和 PCM 混合存储架构, 并能够支持不同形式的混合主存系统模拟, 具有高可配置性. 最后, 论文通过一个使用示例说明了混合主存模拟器编程接口的易用性.

关键词: 相变存储器; 混合主存系统; 模拟器

引用格式: 张德志,万寿红,岳丽华.基于 DRAM 和 PCM 的混合主存模拟器.计算机系统应用,2017,26(9):16–23. http://www.c-s-a.org.cn/1003-3254/5967.html

DRAM/PCM-Based Hybrid Memory Simulator

ZHANG De-Zhi, WAN Shou-Hong, YUE Li-Hua

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China)

Abstract: Phase Change Memory (PCM) has become a candidate of future main memories due to its attractive characteristics of non-volatility, high access speed, and low power consumption. Meanwhile, how to efficiently integrate PCM into current memory systems is becoming a hot topic. Generally, there are a number of choices to use PCM as main memory, e.g., to construct PCM-only main memory systems, or to construct DRAM/PCM-based hybrid memory systems. However, the conflict between numerous PCM-related researches and lack of real devices hinders evaluations of PCM-aware algorithms. Therefore, in this paper, we propose a DRAM/PCM-based hybrid memory simulator. The new features of the simulator are manifold. First, it can simulate different DRAM/PCM-based memory systems, including the hierarchical architecture (DRAM as the cache of PCM) and the hybrid architecture (both DRAM and PCM as main memory). Second, it leverages a clock-accurate timing model to emulate accesses on PCM. Third, it offers a hybrid memory allocation interface that can be easily used by programmers. After a description of the simulator framework, we present basic evaluation results and a case study of the simulator, which suggest its feasibility.

Key words: phase change memory; hybrid memory system; simulator

1　引言

相变存储器(PCM)是一种新型主存技术, 具有读取速度快、静态功耗低、存储密度大以及非易失存储等优良特性[1]. 这些特性使得 PCM 有望取代以动态随机存储器(DRAM)为代表的现有内存技术, 成为未来主存的担当[2]. 另一方面, PCM 由于其材料本身的缘故, 具有写寿命较短、写延迟较长以及写能耗较大等缺陷[3]. 因此, 今后主存系统最有可能的发展方向应该是利用

① 基金项目: 国家自然科学基金(61472376)
收稿时间: 2016-12-30; 采用时间: 2017-01-23

DRAM 和 PCM 优劣互补来构建存储速度更快、存储容量更大、能量效率更高的混合主存系统[4].

然而, 目前仅有为数不多的 PCM 原型芯片问世, 而基于 DRAM 和 PCM 的混合主存系统更是处于研究阶段. 可想而知, 在真实的 PCM 设备上或者基于 DRAM 和 PCM 的混合主存系统中进行 PCM 相关算法的研究及验证是不可行的. 所以目前相关领域的研究工作都是基于模拟平台进行的. 现有的主存模拟器, 如 DRAMSim[5]、NVMain[6], 虽然可以构成混合主存系统, 但是不具备用于混合主存分配的编程接口. 换句话说, 研究者无法按照自己所设计算法的需求, 利用编程语言, 在某种指定的存储介质(DRAM 或者 PCM)上申请或释放自定义大小的存储空间. 因此, 现有的主存模拟器并不能适用于一些主存算法研究(例如页面迁移算法、混合主存索引等)的仿真与验证.

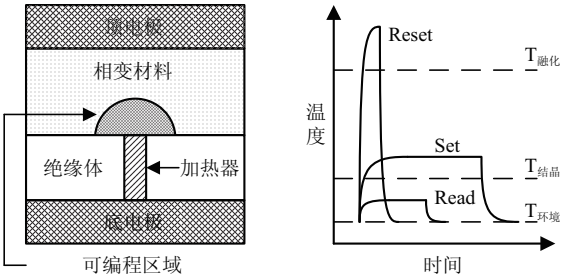
在本文中, 我们构建了一个基于 DRAM 和 PCM 的混合主存模拟器. 一方面, 它有着与现有模拟器相当的性能, 如高可配置性、时钟精确的时序模型以及丰富的仿真数据统计; 另一方面, 它提供了一个友好的用于混合主存分配的用户接口, 极大方便了上述主存算法的仿真与验证工作.

本文余下部分组织如下. 第二部分扼要介绍了相变存储器、混合主存架构以及现有主存模拟器的特点和适用范围; 第三部分描述了所构建的混合主存模拟器的组成结构及其实现细节; 在第四部分, 我们给出了具体的实验及评估, 以说明所构建模拟器的有效性及易用性; 最后在文末给出了全文总结, 并提出今后的工作方向.

2 相关工作

2.1 相变存储器及混合主存架构

图 1(a)所示为一个 PCM 存储单元的基本结构[1]. 不同于传统的主存技术利用电荷存储数据, PCM 是借助相变材料——硫族化合物的关键特性——在晶态与非晶态下巨大的导电性差异来存储数据的. 具体来说就是, 结晶态具有低电阻率, 可标记为 Set 或者“1”状态; 非结晶体则具有相当高的电阻率, 可作为 Reset 或者“0”状态. 在 PCM 上进行 Read、Set 以及 Reset 操作时, 其温度与时间的关系如图 1(b)所示. 需要强调的是, DRAM 和 PCM 在读写实现原理上的差异, 将是我们接下来在构建不同类型主存的时序模型时的重要参考依据.



(a) PCM存储单元的结构图 (b) Read、Set和Reset操作的温度-时间关系变化图

图 1 PCM 存储单元结构图及其上操作的温度-时间关系图

当前, 已提出的混合主存体系结构主要有 2 类: 层次型混合主存[7]和平行型混合主存[8]. 前者是将 PCM 作为主存, 而将 DRAM 作为 PCM 的上一级缓存. 后者则是把 DRAM 和 PCM 作为同级结构整合到一个系统中. 本文所构建的模拟器将尝试实现目前存在的各种混合主存架构.

2.2 主存模拟器

目前已有的主存模拟器主要有 DRAMSim、NVMain、PCRAMsim[9]以及 SIM-PCM[10]等.

DRAMSim 是一个高度可配置的、时钟精确的主存模拟工具, 能够模拟出各种类型的 DRAM, 给出所需的关键性能, 如带宽、延时以及功耗等. 但是它仅仅具备 DRAM 的模型, 并不能对 PCM 的特性进行模拟. NVMain 亦具有高可配置以及时钟精确的优良仿真特性. 此外, 它既能够模拟 DRAM 的运行性能, 还支持对包括 PCM 在内的一系列非易失存储器进行仿真. 但是它只能构建出平行型混合主存系统, 而且主存空间对于外部访问来说是一个整体, 无法对某一种特定的主存介质进行访问. PCRAMsim 则是一个电路级的 PCM 模拟器, 主要用于 PCM 芯片的设计优化或者评估其性能、能耗及器件大小的, 仅仅适用于相关的硬件研究. SIM-PCM 是基于一个全系统模拟器 Simics[11]实现的, 它将每一次主存访问的执行时间设为固定值, 而且忽略了主存内部运行机制对整个系统的影响, 这些都大大降低了仿真结果的准确性和有效性.

3 混合主存模拟器架构

图 2 所示为本文所构建的基于 DRAM 和 PCM 的混合主存模拟器的基本架构. 该模拟器构建了处理器、高速缓存和主存三大层次模块. *CPU* 模块接收由 Pin[12]工具收集到的应用程序的指令序列集, 并将每个

指令转化为一条主存读取操作和若干主存写入操作. 所有这些生成的操作请求都会进入 *Cache* 模块, 该模块是利用一个队列来实现高速缓存的命中(hit)操作和不命中(miss)操作. 主存部分则由主存模块实现, 该模块使得用户可以构建不同结构的主存系统, 包括纯 DRAM 的主存系统、纯 PCM 的主存系统以及基于 DRAM 和 PCM 的混合主存系统. 主存模块从上一层级接收请求, 并在一个具体的主存系统中模拟这些请求的执行情况.

值得一提的是, 该模拟器提供了一套内存分配函数(即 *Dmalloc*()与 *Dfree*(), *Pmalloc*()与 *Pfree*())使得用户可以自主地从 DRAM 或者 PCM 上按需申请或者释放主存空间. 提供这种用于混合主存分配的接口函数的目的就是将主存空间, 特别是存储介质不同这一特性, 展现给编程者. 通过这些接口, 研究者便可以根据他们的主存管理算法来决定将一个读取或者写入请求放入某种特定类型的主存区域.

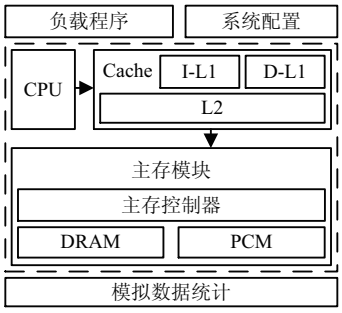


图 2 混合主存模拟器的架构

如图 3 所示, 该模拟器的主存模块是由三个子模块组成, 分别是: 功能接口模块, 主存控制器模块以及主存设备模块. 功能接口模块负责接收访问请求、接收控制器与设备的配置信息, 并将相应的信息发送至主存控制器模块以及主存设备模块中. 主存控制器模块根据功能接口模块的配置信息进行构建, 并处理发送来的访问请求, 将其转化为相应的主存命令序列发送至相应的主存设备中. 主存设备模块同样根据功能接口模块的配置信息进行构建, 同时接收和处理控制器发来的主存命令序列, 最终得到一系列模拟结果. 下面将详细阐述这些模块的具体实现.

3.1 功能接口模块

功能接口模块将整个主存模块的内部运行机制封装起来, 并为用户提供统一的接口, 让他们能够按需定制和使用混合主存系统. 如图 3 所示, 该模块包括外部接口, 主存控制器接口以及主存设备接口.

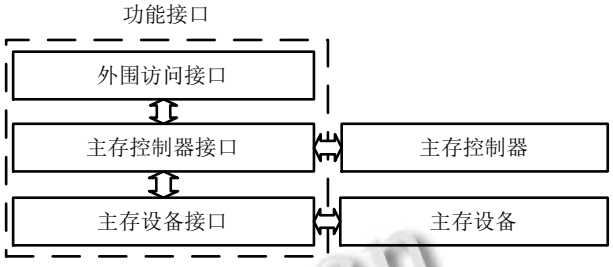


图 3 模拟器主存模块的构成

外围访问接口提供了一个类似总线的数据结构, 其中包含了若干个条目(entry). 移入的请求是以无序的方式放置在这些条目当中, 而且它们均来自应用程序或者上一层级的设备. 这些请求都是以 ID 号、状态、时间戳、访问地址以及访问类型进行标识. 一旦模拟器执行并完成一个请求, 相应的条目就会被清空并置于可用状态. 特别需要指出, 鉴于条目的数量有限, 如果所有的条目都被请求占据, 那么后来的请求在有条目可用之前都将被阻塞搁置.

控制器接口和设备接口则支持对模拟器各种参数的配置. 例如主存控制器方案、主存阵列组织结构以及主存设备时序规格的设置均可通过配置文件来实现. 此外, 性能统计数据, 例如运行时间、各存储区域的延迟分别、总线占用情况以及存储资源使用信息等, 均可按照需求来收集得到.

3.2 主存控制器模块

鉴于目前对主存控制器的内部结构没有统一的标准, 我们尝试根据文献[13]来构建一个通用的主存控制器模型.

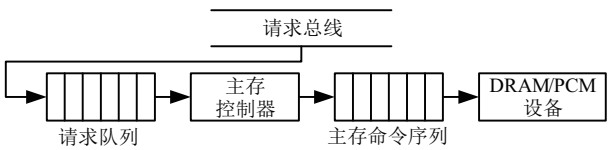


图 4 主存控制器模拟机制的示例

如图 4 所示, 主存控制器模块首先根据请求排序策略从总线中选择请求放置到请求队列中. 接着, 请求队列根据地址映射方案将请求的物理地址映射到主存片上位置. 然后, 基于控制器使用的行缓冲(*Row Buffer*)管理机制, 将每个请求转化成一组相对应的主

存命令序列(如表 1 所示). 最后将这些主存命令发送到指定的主存存储单元之中.

表 1 主存命令及其描述

命令	描述
行访问命令(RAC)	从存储单元中取一整行数据放入行缓冲中
列读取命令(CRC)	将数据通过共享数据总线返回到主存控制器
列写入命令(CWC)	将数据从行缓冲中写入主存单元之中
行预充电命令(RPC)	为另一条行访问命令重置行缓冲区

当主存设备的行缓冲策略被设置为开放模式(open policy)时, 行缓冲中存储的一整行数据将会持续存在, 以便随时提供给下一次访问使用. 在此策略下, 一个主存访问请求根据不同的情况有三种可能的转变. 一是, 如果新来的访问与之前完成的访问发生冲突, 即需要提取不同的整行数据放入行缓冲中, 那么该条请求将会被转化为一条行预充电命令(Row Pre-charge Command, RPC)、一条行访问命令(Row Access Command, RAC)和一条列访问命令(Column Access Command, CAC). 二是, 如果当前存储单元处于空闲状态, 即没有访问在该区域进行操作, 那么一个请求将相应地转化为一条 RAC 和一条 CAC. 三是, 如果行缓冲存储的行数据正是所需的那一行数据, 那么该请求将仅仅转化为一条 CAC. 需要指出, 列访问命令究竟是读取还是写入, 则完全取决于其请求类型是读还是写. 与上述描述相反的是, 当行缓冲策略是关闭模式(close policy)时, 每一次访问完成后均会直接重置行缓冲, 以便为接下来的访问做好准备. 这种策略下, 每一个访问请求均会被转化为相同的主存命令序列, 依次为: 一条 RAC, 一条 CAC 和一条 RPC. 本文中, 行缓冲策略默认采用开放模式.

3.3 主存设备模块

3.3.1 主存阵列组织

图 5 给出了模拟器所模拟的存储器阵列布局. 通常来说, 一个主存阵列由若干个独立的 *Rank* 构成, 它是主存设备最顶层的结构, 是具有全功能的主存单元, 能够被独立地访问. 而多个 *Bank* 构成一个 *Rank*, 它们同时操作以满足内存操作. 此外, 每个 *Bank* 由若干个 *Subarray* 组成, 它们是主存最为基础的存储单元.

3.3.2 时序模型

我们利用一系列的时序参数来表征上面提到的那些基本主存命令的执行, 以此实现一个动态的、细粒度的、时钟精确的时序模型. 表 2 枚举了在时序模型中需要使用的所有参数及其含义. 鉴于该模拟器建立的 DRAM 的时序模型与 DRAMSim 的相似, 故在此不再赘述. 不同于 DRAM 的读写机制, PCM 在读取数据后, 不需要将数据再重新写回存储介质上; 另一方面, PCM 上没有刷新操作. 因此, 接下来我们便对主存中 PCM 命令执行时间的测量方法进行详细描述.

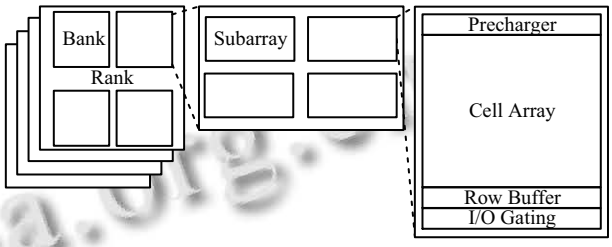


图 5 模拟器构建的存储器层次结构

表 2 主存时序参数汇总

符号	描述
t_{Burst}	将数据经数据总线放到控制器所需的时间
t_{CAD}	将数据从行缓冲放入*I/O Gating*的时间
t_{DCD}	列读取访问与从主存中取出数据的时间间隔
t_{CMD}	命令从控制器经由总线传至主存设备的时间
t_{CWD}	将列写命令与数据放到相应总线的时间
t_{RCD}	将数据放入行缓冲所需的时间
t_{RP}	将行缓冲清空重置所需的时间
t_{RRD}	对同一存储单元不同行访问, 行间转换时间
t_{WP}	使相变材料改变状态的写脉冲的持续时间
t_{RAS}	行访问指令与数据重写DRAM的时间间隔
t_{WR}	将数据从行缓冲写回DRAM所需的时间
t_{RFC}	DRAM中刷新操作执行时间

1) RAC 的时序模型

RAC 命令是访问某个数据之前, 对该数据所在的一整行的数据进行定位和准备. 其作用便是告知主存设备把即将访问的数据准备好, 放在行缓冲中, 等待接下来的指令从该行数据中存取指定的数据. RAC 的执行过程如下所述: 首先, 主存系统需要时间将行访问命令从控制器传送至主存设备中(t_{CMD}). 接着将数据从存储单元放入行缓冲之中(t_{RCD}). 此外, 考虑到如果行缓冲策略是开放模式, 那么两条 RAC 访问同一个存储单元所造成的冲突延迟(t_{RRD})就需要考虑进来. 因此, RAC 的执行时间可以用表达式(1)来刻画.

$$t_{RAC} = t_{CMD}(+t_{RRD}) + t_{RCD} \quad (1)$$

2) CRC 的时序模型

CRC 命令是读请求的核心部分. 它是从 RAC 所定

位和准备的一行数据中, 取出所需要的数据, 并通过共享数据总线返回至控制器中, 从而实现一个基本的读操作. 对于 CRC 的执行, 首先是将所请求的数据从行缓冲区放入 *I/O Gating* 上. 然后把数据经过数据总线传到主存控制器中(t_{Burst}). 另外考虑到上述两个操作执行之间的时间间隔(t_{DCD}), 我们给出 CAC 执行时间的最终表达式(2).

$$t_{CRC} = t_{CAD} + t_{DCD} + t_{Burst} \tag{2}$$

3) CWC 的时序模型

CWC 命令则是写请求的核心部分. 对于它即将写入的新数据, RAC 命令已经将新数据对应的一整行数据放入相应的行缓冲之中, CWC 将更改行缓冲之中的旧数据为所需的新数据, 从而完成一个写操作. 首先, 主存系统需要将访问地址和列写入命令分别放到地址总线和命令总线上. 由于这两个操作可以同时执行, 因而这个过程所花费的时间便是控制器将命令和数据安置好所需的时间(t_{CWD}). 接下来, 数据流从数据总线上通过进出门, 存放到行缓冲之中(t_{Burst}). 最终便将最新接收到的数据从行缓冲写入存储单元之中(t_{WP}). 综上所述, 整个 CWC 的执行时间便是上述三者之和, 如表达式(3)所示.

$$t_{CWC} = t_{CWD} + t_{Burst} + t_{WP} \tag{3}$$

4) RPC 的时序模型

RPC 命令的作用是为接下来的行访问命令重置行缓冲, 因而找到对应的行缓冲, 再将其中的数据清空即可. 它执行过程便为如下两步: 第一步, 主存系统将一条 RPC 命令发送到主存设备上(t_{CMD}). 第二步, 相应的 *Subarray* 上的行缓冲执行预充电操作, 即清空行缓冲中的数据, 为接下来的访问请求做好准备(t_{RP}). 所以, 我们用表达式(4)来描述一条 RPC 命令执行时间.

$$t_{RPC} = t_{CMD} + t_{RP} \tag{4}$$

4 实验及评估

本部分, 我们首先将所构建模拟器与其他现有模拟器进行对比, 以验证其时序模型的正确性. 其次, 通过构建两个目前常见的混合主存架构, 即层次型混合主存系统和平行型混合主存系统, 来说明模拟器的高可配置性. 最后, 利用一个使用示例说明如何利用混合内存分配接口来实现用户可控的混合主存管理算法的实验.

4.1 实验配置

为了实现一个准确的、有效的、可比的模拟实验, 首先需要统一模拟系统中的处理器、高速缓存以及主存的实验参数. 表 3 列出了实验中所需的处理器以及高速缓存的配置信息.

表 3 系统配置信息

CPU	L1 Cache	L2 Cache
2 GHz	32 kB l1_icache 32 kB l1_dcache 64 Bytes cache-line	2 MB 64 Bytes cache-line

至于主存的阵列组织, 我们依据实际芯片的数据表来设置 DRAM[14]和 PCM[15]的参数. 表 4 涵盖了主存配置文件中涉及到的各种参数值, 从中可以看出 DRAM 和 PCM 的显著差异. 此外, 所有实验均采用 SPLASH-2[16]基准测试集的一部分作为测试用例.

表 4 主存时序参数(单位: ns)

参数	DRAM	PCM
t_{Burst}	12	12
t_{CAD}	15	15
t_{DCD}	6	3
t_{CMD}	10.5	10.5
t_{CWD}	12	21
t_{RCD}	13.5	80
t_{RP}	13.5	13.5
t_{RRD}	7.5	7.5
t_{WP}	*n/a*	150
t_{RAS}	36	*n/a*
t_{WR}	15	*n/a*
t_{RFC}	160	*n/a*

4.2 仿真有效性验证

模拟器 DRAM 部分的验证方法是: 在本模拟器和 DRAMSim 上运用相同的测试用例、配置参数进行模拟, 对比分析它们的运行结果. 图 6(a)显示了 SPLASH-2 的部分测试程序运行在模拟的三星 DRAM 芯片[14]上的平均指令时钟数(Cycles Per Instruction, CPI). 可以看到, 我们构建的模拟器与 DRAMSim 之间有着 0.5%~5.1% 的偏差.

另一方面, 对于纯 PCM 的主存系统, 我们以相同的系统配置参数在该模拟器和 NVMain 之间进行对比实验. 这部分实验是基于三星 PCM 芯片[15]的参数进行模拟的, 其实验结果如图 6(b)所示. 与 DRAM 模拟的情况类似, 二者存在 2.6%~6.0% 的不一致.

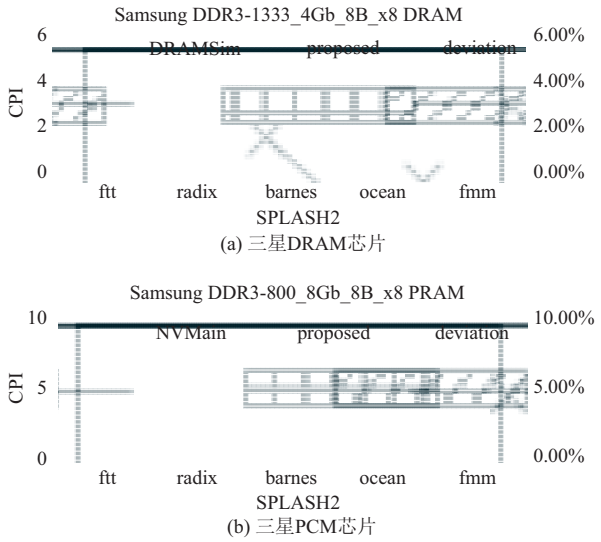


图 6 测试用例在 DRAM 和 PCM 上运行的 CPI

通过上述对比实验, 我们看到上述模拟器的仿真结果仅仅有微小的差异, 基本上是一致的. 鉴于这三个模拟器的主存控制器在设计上是不同的, 而它们的时序模型所采用的原理是相似的, 因此我们认为上述实验结果的微小偏差是它们主存控制器设计差异所导致. 此外, 由于 DRAMSim 和 NVMain 在文献[5,6]中已经得到有效性验证, 再结合对比实验的结果, 我们可以认为所构建模拟器的仿真结果是有效的、可信的.

4.3 混合主存系统构建

这部分实验的目的是说明该模拟器能够搭建各种混合存储架构来实现仿真模拟; 同时, 说明模拟器的高可配置性. 下面依次构建了两类常见的混合主存架构, 即层次型(DRAM 作为 PCM 的上一级缓存)和平行型(DRAM 和 PCM 作为同级主存)混合主存系统.

首先, 我们实现了层次型主存系统, 即把 DRAM 作为 L-3 cache. DRAM 大小分别设置为 32 MB、64 MB、128 MB 以及 256 MB, 同时将 PCM 的容量固定为 8 GB. 实验统计数据如图 7 所示: DRAM 的大小与测试用例的 CPI 呈现出反比关系. 考虑到写请求所造成的影响, 程序 *barnes* 和 *ocean*(二者的写请求比例较高)的 CPI(其越小说明运行速度越快)下降得更明显一些.

对于构建平行型主存系统的实验, 我们将主存(总容量为 8 GB)全为 DRAM 的结构作为基准系统. 接下来, 通过修改实验配置, 实现 4 个混合主存系统, 依次是: 6 GB DRAM + 2 GB PCM, 4 GB DRAM + 4 GB PCM, 2 GB DRAM + 6 GB PCM 以及 8 GB PCM. 考虑到 PCM 上的写延迟更长, 可以推测, 随着程序中写比例增长以及 PCM 在混合主存中的比重的增加, 同一测试程序的模拟运行时间也应该增加. 而通过实验, 我们测得这些测试用例(*fft*, *radix*, *barnes*, *ocean*, *fmm*)的写请求比例分别为 0%、10%、60%、63%、66%. 如图 8 所示, 写比例低的 *fft* 和 *radix*, 随着 PCM 比重的增加, 它们的运行速度减缓的并不多. 而对于其余三个写比例高的测试用例, 它们的运行速度与 PCM 比重程序显著的反比关系. 这些实验结果与预期分析是相符的.

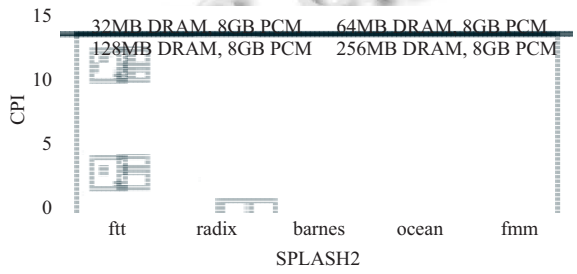


图 7 测试用例在层次型主存系统上运行的 CPI

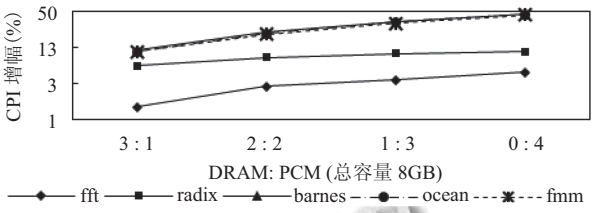


图 8 测试用例在平行型主存系统上运行的 CPI

通过两类混合主存系统的构建, 充分说明了该模拟器的可配置性很好, 能够模拟出各类主存架构系统, 以满足不同研究的实验需求.

4.4 使用示例

本节意在说明如下两点: 一是混合主存分配接口具有简单的可操作性, 二是将该模拟器运用于混合主存管理的研究是十分有用的.

目前, 所有现有的模拟工具均不能将混合主存空间直接暴露给编程者. 更形象地说就是, 用户使用主存空间时, 只能够将主存系统视为一个整体, 并不能够知道其中哪一块存储空间是 DRAM 设备的, 哪一块空间是 PCM 设备的. 而这个问题对于主存算法的研究者来说是十分棘手的: 他们不能够根据自己的算法在指定的内存设备上开辟一定大小的存储空间. 本模拟器便提供了一个混合主存分配的接口, 以实现友好的、用户可控的主存管理.

在此, 我们以混合主存系统中常见的页面迁移机制的研究为例, 说明如何在主存研究中实验该模拟器. 通常, 考虑到 PCM 的写速度较慢以及它的写寿命较短这两大缺陷, 为了提高混合主存系统的运行效率, 研究者最基本的思路便是尽可能地让写操作在 DRAM 设备中执行. 基于这个思路, 我们设计了一个朴素的页面迁移机制: 通过一个固定的概率来决定是否对一个将要分配在 PCM 上的页面进行迁移. 由此, PCM 上的写操作可以得到一定程度的降低, 从而达到改善混合主存系统运行效率的目的. 具体来说就是, 当一个请求调用了 *Pmalloc*()函数, 进而在 PCM 上申请主存空间时, 我们利用一个具有指定概率的随机发生器得到一个布尔值. 如果该值为真, 那么便将该条请求从 PCM 迁移到 DRAM 上执行, 反之则按原样执行. 例如, 若要在 PCM 上为 100 个 *size_t* 型数据分配存储空间, 那么根据上述算法, 其伪代码如下所示:

```
size_t *p;
p = (size_t *) Pmalloc (sizeof (size_t) * 100);
if(bool_random(possibility) == false)
/* bool_random 有 possibility 的概率返回 true,
                有 1-possibility 的概率返回 false*/
{
    Pfree(p);
    p = (size_t *) Dmalloc (sizeof (size_t) * 100);
}
```

我们在一个平行型混合主存系统(2GB DRAM + 6GB PCM, 其中 PCM 有 3 个 channel, 每个 channel 有 2GB 的 PCM)上对这个策略进行评估. 由于 SPLASH-2 的测试程序并未考虑在混合主存的不同类型主存中进行内存空间分配, 因此我们调用 1 个 *Dmalloc*()和 3 个 *Pmalloc*()代替原有的主存空间申请函数, 而每个函数申请的空间大小是相同的.

图 9 的实验结果显示: 在使用这个简单的页面迁移机制的情况下, 当迁移概率从 0% 改变至 75% 时, 测试程序的 CPI 改善了 4% 左右. 与此同时, PCM 上的写操作减少了大约 40%. 可以说, 通过实验, 我们验证了所提策略对于提高混合主存运行速度以及减少 PCM 上写操作的推理. 然而从实验结果中也可以明显地看出, 这个简单迁移机制的弊端便是增加了系统的请求指令执行总数, 特别是 DRAM 上请求指令执行次数的增加, 必将使得 DRAM 成为整个系统运行的瓶颈.

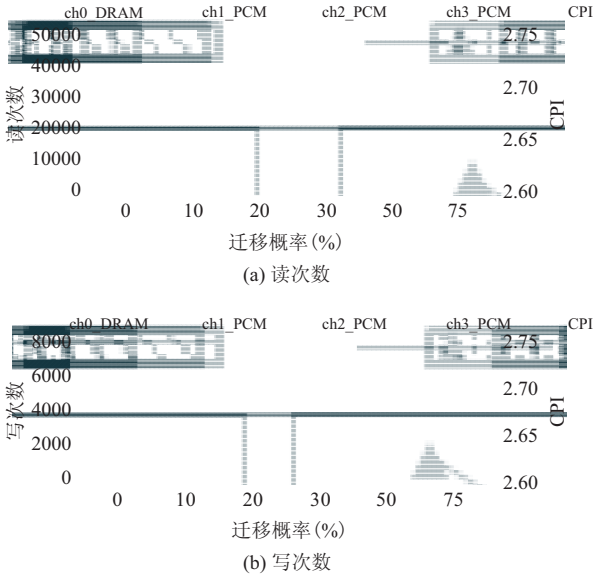


图 9 不同迁移概率下的读写次数以及 CPI 统计(测试用例: 修改的 *radix*, 2 GB DRAM, 6 GB PCM)

最后需要强调的是, 该使用示例并不是试图提出一个优越的页面迁移算法, 更不是证明所提的简单算法的优越性. 其重点是为了说明研究者可以很容易地将该模拟器运用于主存算法研究的验证评估实验当中.

5 结语

在本文中, 我们基于 DRAM 和 PCM, 设计并实现了一个有效的、可配置的、易用的混合主存模拟器. 我们描述了模拟器的整体架构及关键技术, 详述了在时序模型的有效性验证以及构建多种存储架构的可行性方面所做的工作, 探讨了如何运用该模拟器进行仿真模拟实验, 实现了用户可控的混合主存分配接口, 为混合主存的相关研究提供了一个统一便捷的实验工具.

众所周知, 目前基于 DRAM 和 PCM 的混合主存有着广阔的技术前景, 它们为日后实现大容量、低能耗、高可靠的主存系统带来了希望. 而在大数据时代下, 如何提高系统的能耗有效性正逐渐成为混合主存领域的研究热点. 因此, 我们今后的工作将是在不断完善该主存模拟工具的基础上, 将能耗仿真模型整合其中, 以满足更广泛研究的使用需求.

参考文献

1 Wong HSP, Raoux S, Kim S, *et al*. Phase change memory.

Proc. of the IEEE, 2010, 98(12): 2201–2227. [doi: 10.1109/JPROC.2010.2070050]

2 Chen KM, Jin PQ, Yue LH. A novel page replacement algorithm for the hybrid memory architecture involving PCM and DRAM. IFIP International Conference on Network and Parallel Computing. Berlin Heidelberg, Germany. 2014. 108–119.

3 Li L, Jin PQ, Yang CC, *et al*. Optimizing B+-tree for PCM-based hybrid memory. Proc. of the 19th International Conference on Extending Database Technolog (EDBT). Bordeaux, France. 2016. 662–663.

4 Wu ZL, Jin PQ, Yue LH. Efficient space management and wear leveling for PCM-based storage systems. Proc. of the 15th International Conference on Algorithms and Architectures for Parallel Processing. Berlin Heidelberg, Germany. 2015. 784–798.

5 Rosenfeld P, Cooper-Balis E, Jacob B. DRAMSim2: A cycle accurate memory system simulator. IEEE Computer Architecture Letters, 2011, 10(1): 16–19. [doi: 10.1109/L-CA.2011.4]

6 Poremba M, Zhang T, Xie Y. NVMain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems. IEEE Computer Architecture Letters, 2015, 14(2): 140–143. [doi: 10.1109/LCA.2015.2402435]

7 Qureshi MK, Srinivasan V, Rivers JA. Scalable high performance main memory system using phase-change memory technology. ACM SIGARCH Computer Architecture News, 2009, 37(3): 24–33. [doi: 10.1145/1555815]

8 Kim D, Lee S, Chung J, *et al*. Hybrid DRAM/PRAM-based main memory for single-chip CPU/GPU. Proc. of the 49th ACM/EDAC/IEEE Design Automation Conference (DAC). San Francisco, CA, USA. 2012. 888–896.

9 Dong XY, Jouppi NP, Xie Y. PCRAMsim: System-level performance, energy, and area modeling for phase-change RAM. Proc. of the 2009 IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers. San Jose, CA, USA. 2009. 269–275.

10 Li X, Lu K, Zhou X. SIM-PCM: A PCM simulator based on simics. Proc. of the 4th International Conference on Computational and Information Sciences (ICCIS). Chongqing, China. 2012. 1236–1239.

11 Magnusson PS, Christensson M, Eskilson J, *et al*. Simics: A full system simulation platform. Computer, 2002, 35(2): 50–58. [doi: 10.1109/2.982916]

12 Luk CK, Cohn R, Muth R, *et al*. Pin: Building customized program analysis tools with dynamic instrumentation. ACM SIGPLAN Notices, 2005, 40(6): 190–200. [doi: 10.1145/1064978]

13 Aboulenein N, Osborne RB, Huggahalli R, *et al*. Method and apparatus for memory access scheduling to reduce memory access latency: U.S. Patent 6785793. 2004-08-31.

14 Moon Y, Cho YH, Lee HB, *et al*. 1.2V 1.6Gb/s 56nm 6F2 4Gb DDR3 SDRAM with hybrid-I/O sense amplifier and segmented sub-array architecture. Proc. of IEEE International Solid-State Circuits Conference Digest of Technical Papers. San Francisco, CA, USA. 2009. 128–129.

15 Choi Y, Song I, Park MH, *et al*. A 20nm 1.8V 8GB PRAM with 40MB/s program bandwidth. Proc. of the 2012 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC). San Francisco, CA, USA. 2012. 46–48.

16 Woo SC, Ohara M, Torrie E, *et al*. The SPLASH-2 programs: Characterization and methodological considerations. ACM SIGARCH Computer Architecture News, 1995, 23(2): 24–36. [doi: 10.1145/225830]