Page Coloring的历史与发展

刘磊（Sys-Inventor实验室）

历史背景

在大约1985年前后，MIPS操作系统采用了“页着色”（Page Coloring）技术，这是有文献可查的第一次正式出现的“页着色”这一名词 [1,2,3] 。在那个时代，“分时”和“多任务”的设计思想已经使得系统的整体吞吐量和服务能力有所提高。但是，系统设计者却发现，由于从虚页到实页的“任意映射”（Arbitrary Mapping），数据块在采用物理地址索引的Cache中的分布通常是不均匀的。这就导致了系统性能通常会产生很大的波动，即使是同一道程序的多次运行，执行时间也有可能会发生明显的变化。为了解决这个问题，研究人员者设计了新的页面虚实转换的算法：在映射过程中，让物理页面的“低几位”（low order bits）与虚拟页面的“低几位”满足一定的函数关系（如，相等、固定偏移距离，等等）。在这种情况下，Cache中的数据分布就不再会出现明显不均匀的情况了，因为虚页的映射不再是随机、任意和无序的。我们可以想像，在当时研究人员的脑海里，以若干个地址位为依据，将不同的虚、实页面着上了不同的“颜色”（比如，低三位为000的页面为红色，001的页面为蓝色，等等），那么整个内存系统就变的五颜六色、生动活泼了起来。不能不说Page Coloring的诞生是贴切和富有创造力的，这就是Page Coloring的最初的形态。在SUN的早期版本的UNIX操作系统里就采用了Page Coloring技术。上世纪90年代，Page Coloring的优势还常常体现在降低地址转换时候的硬件开销。因为在采用Coloring的机器上，虚拟页和物理页在若干个低位上已经建立了对应关系，因此，这种设计减轻了负责地址转换的MMU的设计开销。

1990年代是Page Coloring技术发展的一个上升期，很多研究组都致力于这项技术的优化与改进。基于Page Coloring的一个著名的研究工作是“Best Bin”(Bin Hopping) [2] ，由Richard E.Kessler和Mark D. Hill等科学家提出，论文收录于1992年的TOCS。他们发现了传统的Page Coloring技术有可能会引发不同地址空间（如Stack，Heap，Text）之间在某一组Cache Set(Cache Bin)中的竞争。那么，有可能会发生的糟糕的情况是，频繁被访问的虚拟页被映射到了同一个Bin中，进而造成Cache的颠簸。Bin Hopping改进了Page Coloring，该算法以Bin为映射单位，将连续的虚拟页面映射到连续的Bin之中，因此有效的避免了上述问题。

工业界也对Page Coloring技术显示了浓厚的兴趣。无论是从地址映射开销的角度出发，还是从提高Cache使用率的观点切入，工业界都做出了足够多的尝试。例如，VMware考虑到Guest OS可能使用Page Coloring技术，因此，在虚拟机监控系统ESX Hypervisor中就曾经包含了提供Page Coloring优化的兼容版本，在为多个Guest OS分配物理页面的时候采用Page Coloring技术来匹配Guest OS所申请页面的颜色。换言之，Hypervisor会尽可能满足Guest OS对颜色的需求，以确保最大限度保留客户系统的优化手段。

发展

2000年后计算机系统逐步进入多核时代。在享受多个计算单元带来的高吞吐量的同时，人们又必须忍受并发多线程在宝贵的Cache资源上的相互竞争与干扰［7,11］。业界越来越认识到，共享的最后一级Cache(Last Level Cache,LLC)在多核时代将是制约计算机整体性能的非常严重的问题,有的科学家曾严肃的称为“The trouble with multicore”，还有科学家甚至认为“Multicore is bad news for supercomputers”［11］。

针对这一问题，研究人员再次诉诸于Page Coloring。只是这一次应用场景与解决问题的方法都和80、90年代不同了。具体的做法是，如图1所示，以Cache Set的物理索引位为依据，将Cache Set划分为不同的颜色组，并在页面虚实转换的过程中，将不同的颜色分配给不同的进程（线程），那么每一个进程仅使用属于自己颜色的一组Cache Set，进程之间在同一组Cache Set上的竞争也就不复存在了。这种方法思路简单，可操作性很强，并且易于部署。可以说在进程（线程）间互扰严重的情况下，该方法的效果立竿见影，而开发成本很低，仅需要修改操作系统的资源分配模块（大约1500行代码）［7］。

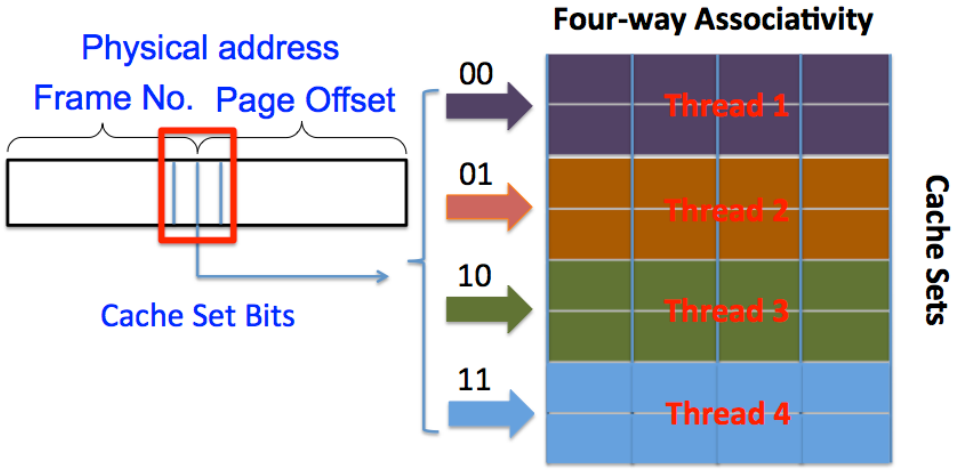

图1. 基于Page Coloring技术Cache划分

Page Coloring是纯软件的方法，因此更容易在业界得到普及。在使用Linux操作系统的计算机上，为使用Page Coloring技术修改Buddy System内存管理模块是相对容易的。就学术界而言，很多研究工作不但探讨了Page Coloring的有效性与工作集访存特点之间的关系，还研究了在运行时如何动态的调整“颜色”的分配，以达到最佳性能的目的[3,7]。工业界对这方面的研究成果做出了积极的响应,根据Intel的一份公开文件[11]，在论文[7]中发表的LLC划分的软件技术已在多核计算机的生产系统中被有效地采纳，通过这项技术获得了减少1.5倍延迟的效果，并将相关成果应用于其他企业客户的生产线。另外,不仅在优化系统吞吐量上的表现可圈可点，在保证系统服务质量（QoS）方面，Page Coloring技术也有相当的潜力可被挖掘，因为通过保证关键程序的Cache资源使用量,并将其与可能会产生潜在干扰的程序隔离开，就有可能提供较高的服务质量。

从本质上来讲，Page Coloring实际是调整了数据在物理存储介质中的排布，其根源是资源管理与分配的方式。其扩展形式也有很多应用和研究场景，例如，［6］

提出的通过操作系统级的页面分配机制来管理分布式Cache的方法；［4］设计的结合编译器技术的OS资源分配与Cache划分方法；以及［5］采用软硬结合方式，并基于程序行为来调整页面分布的方法，等等。

综上所述，在2010年之前这段时间内，Page Coloring技术被广泛使用，很多公司逐步形成颇具自身特色的优化手段。除了前文中提到的VMware公司和Intel公司，据笔者所知，SUN公司的操作系统，Oracle公司的数据库管理系统，都不同程度的借鉴或采用过Page Coloring相关的技术路线。

新时代的新问题

线程间在LLC上的竞争不仅仅是多核时代需要面临的唯一问题，在DRAM系统上的访存竞争和干扰也会影响系统性能。目前，计算机的主存（Main Memory）系统是由若干DRAM Bank构成的，在多核体系结构下，这些Bank被运行于多核之上的多个线程所共享。由于以行缓冲为核心部件的的DRAM系统的特殊组织形式，线程间共享将会引发在DRAM Bank上的访存干扰，造成行缓冲的颠簸，增大访存延迟，进而影响系统的整体性能。因此，如何优化在DRAM上的访存行为，是国内外系统结构领域研究的热点问题，很多研究集中在优化内存控制器中的访存调度算法。

目前被工业界广泛采用的内存控制器调度算法还比较简单，而学术界提出的很多较先进的访存优化算法由于种种原因还没有被工业界广泛采用。举例来说，目前被广泛使用的依然是FR-FCFS调度算法。该算法的思想源于First Come First Service (FCFS),即内存控制器优先调度先发出的内存请求。但这个方法的缺陷在于没有考虑到DRAM行缓冲的局部性。FR-FCFS是对FCFS的改进，在保持先来先服务的基础上，在一个时间窗口内调度所有访存序列以尽量命中已经打开的行缓冲。但是这种“单纯”的最大化行缓冲利用率的调度方式固然容易实现但在两个重要的指标公平性（Fairness）和吞吐量（Throughput）表现的却都不是很理想。国内外研究团队对访存优化的研究都展示浓厚的兴趣，各种算法层出不穷，但由于很多State-of-the-Art的调度算法存在着很多不确定的因素而难于被产品化。例如，较为先进的TCM［12］调度策略，不但需要引入额外的4K（最小）的存储空间，还需要通过辅助硬件来分别记录很多关于进程在运行时的信息，并且在调度时也需要复杂的逻辑支持，因此，TCM引入的开销很难被衡量和优化，为产品化的设计与实现带来困难。笔者认为，另一个问题是，即便调度算法在很多情况下是有效的，其机制也很难在根源上彻底消除线程间的访存干扰。

（1）DRAM Bank Partitioning —— Page Coloring在Bank层次的扩展

业界期望功能性强、实用性强的访存优化机制。既然在Cache上有可能通过Page Coloring消除线程间的访存干扰，那么在DRAM上是否也有此可能？如图2所示，理论上，若能将系统所有的DRAM Bank划分成若干组（示以不同颜色），让每个运行的程序（线程）使用且仅使用属于它自身的那一组Bank，那么程序间的访存干扰也就不复存在了，程序自身的行缓冲局部性也得以保证。更为形式化的表示为，这种优化机制将全部的DRAM Bank划分为若干个互相没有交集的子组SUB_i(Bank)，并且，

$U(SUB_i\ (Bank))$= Main Memory Amount。

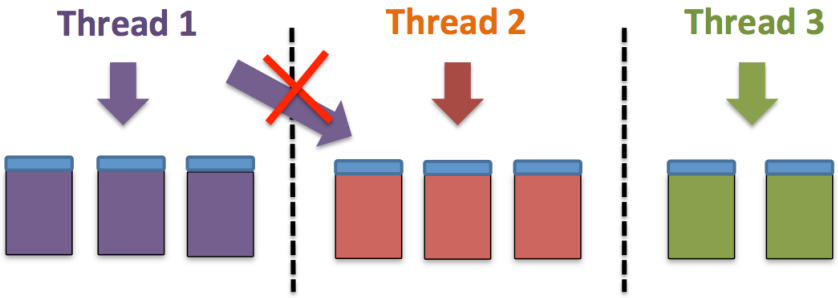

图2. 在DRAM Bank上的“着色”

为此达此目的，学者们再一次诉诸于Page Coloring这项发展了二十多年的技术，尝试将LLC划分的方法扩展到DRAM层。如果能够利用地址映射中的Bank bits，将Bank分为不同的颜色的类别，那么即可达到利用Page Coloring划分Bank的目的。学者们进一步研究了某些典型系统的地址映射（如intel-i7 series），发现其中有几位能够能够索引DRAM Bank [8,14] 。因此，从理论上来讲，将Page Coloring技术应用在DRAM Bank上是可能的，只要按照线程对“颜色”的需求，在分配物理页面时有选择的将属于特定颜色的物理页面分配给特定的线程即可。

在DRAM上应用Page Coloring的做法是简洁有效的。如图3所示，在不采用Bank划分（Bank Partitioning Mech., BPM）的情况下，来自不同程序的访存请求被随机分散到任意Bank上，由于线程间相互干扰引发的行缓冲颠簸将导致部分访存请求的延时较长；但相比较之下，由于采用了DRAM Bank划分技术（w/BPM），消除了程序之间的访存干扰，每个程序的行缓冲局部性都有所提高，因此，访存请求的响应延迟相对减小，有利于系统总体性能。从访存协议和DRAM内部操作等微体系结构的因素来看，Bank划分将有可能直接提高访存请求的响应速率，图3还向我们展示了在使用BPM的情况下（理想情况），每个Bank上的访存请求都节省了近1/3的响应时间，每个线程的运行速率都有可能因此而提高。

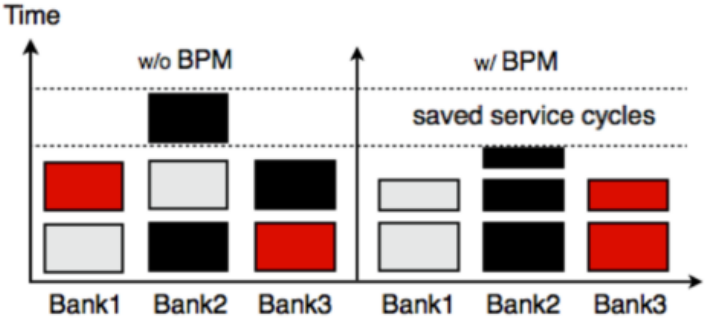

图3. Bank划分的理论效果图

（2）Channel Partitioning —— Page Coloring在通道层次的扩展

研究者后来发现对DRAM系统的划分可以进行的更为彻底，即将Page Coloring技术再扩展到Channel(通道)层次[10]。当前计算机普遍采用多通道(Multi-Channel)技术，目的是为了增加系统的总体带宽。例如，DDR3-1600的峰值带宽是12.8GB/S，如果系统有两条通道，那么系统的整体理论峰值带宽将是25.6GB/S。装备双通道的计算机已经非常普遍，某些高性能的服务器还采用四通道、八通道技术。对于主流

多通道系统，通常采用以Cache Line (64B)为单位的“交替”访存，将访存平均分配在若干个通道内，希望能够以简单的方式充分利用带宽。然而，“盲目”交替访存实际上会加剧在所有通道内对带宽的竞争，同时也可能会引发更多的DRAM上的冲突，因此在很多情况下未必会产生收益。如图4所示，不同颜色的色块代表来自不同程序的访存请求，图中每一个通道均要向所有程序提供服务，再加上Bank中的访存冲突，那么线程间的访存延迟就有可能会更为严重[13]。本文将这种在全部通道、全部Bank上的竞争和干扰理解为all-to-all的DRAM系统上的资源竞争和访存干扰。研究者尝试在结合BPM的基础上，限定任意程序仅访问某一个通道，以避免这种（all-to-all）的访存干扰。如图5所示，研究者期望，在满足内存需求量的情况下，每个程序仅使用某个通道并且在每个通道内部使用Bank划分。因此，在每个通道上的竞争被降低的同时，Bank上的冲突也被消除了，这种机制被命名为BPM+[10]。在实现上，有些平台的BIOS为我们提供了可配置的选项，可以通过调整这个选项来启用或取消“交替”模式。以Intel i7-860为例，在调整BIOS之后，地址最高位即为Channel位。着色的方式与上文提到的方式类似，差别在于分配物理页面时用来索引页面的地址位同时包含了Channel位和Bank位这两部分。

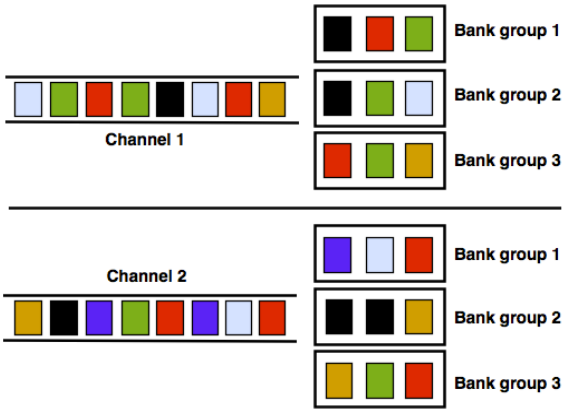

图4. 线程间通道上的访存“交替”

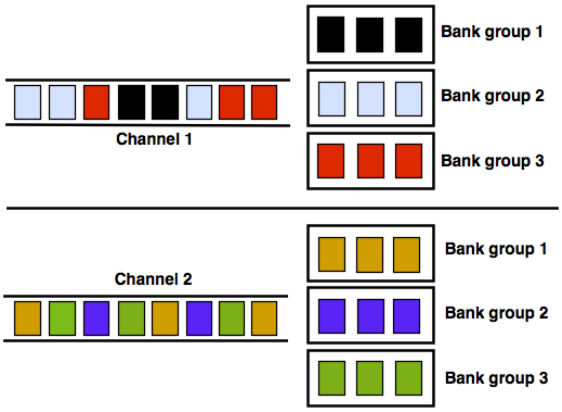

图 5. 在 Channel 上的“着色”

三种在存储器不同层次上的Page Coloring技术对系统整体性能的影响对比如图6所示。对于随机产生的20组由4程序或8程序组成的工作集来说，BPM和BPM+对系统性能平均提升5%左右（最高接近10%），并且BPM+的性能要优于BPM。值得注意的是，Cache划分的效果平均来看不如BPM和BPM+，并且在有些情况下会造成系统性能的下降。

(3)Vertical Partitioning —— 穿越多级存储器层次的Page Coloring

随着Page Coloring技术在内存储器各个层次上的使用，有的学者猜想是不是能将在多个层次上划分技术带来的好处都累加在一起，进而产生更大的性能受益？图7展示了对200多组随机产生的8-programmed或者4-programmed包含SPEC2006程序的工作集使用Page Coloring技术产生的性能效果。对每一组工作集都验证划分Cache (Cache-Only)和划分Bank (Bank-Only)时的有效性。划分的效果随工作集的变化而变化，但总的来讲，在不使用I/O swap的情况，Bank划分在绝大多数的情况下能够带来系统性能的提升，但Cache划分在很多情况下会使系统性能下降，工作集敏感性的特点非常突出。

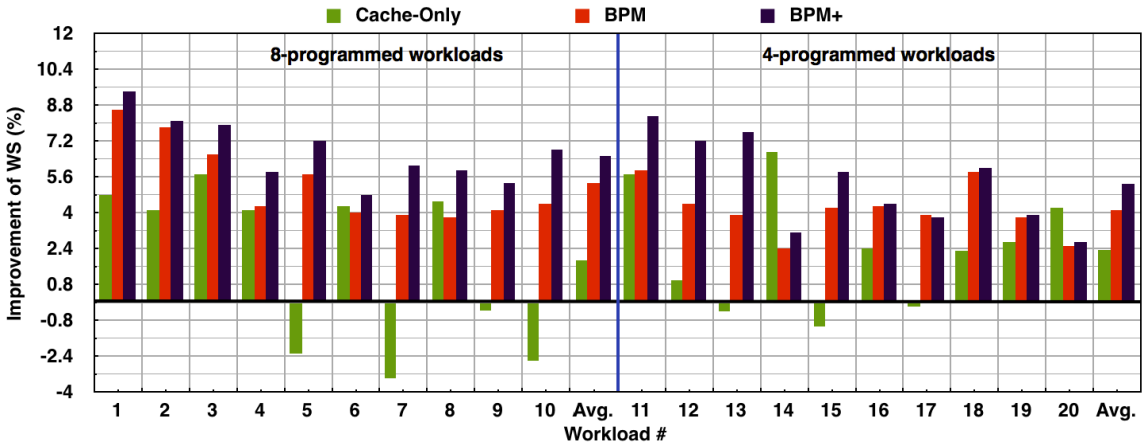

图6. 三种基于Page Coloring的划分机制的性能对比（i7-860, 2.4GHz, 8GB DDR3）

从这个图上还可以观察到，对于第一象限中的工作集，大部分的性能提升仅在2%～4%之间。虽然对于真实的计算环境来说这并不能被忽略，但研究人员还是希望系统性能对于第一象限中的这些工作集可以进一步被提升，回到前面的问题，能不能将多个层次上的划分效果累加在一起？从工业界的角度来讲，通用处理器的速度已接近极限，如果能对近50%的随机工作有增强的优化效果，则这个动机是有价值的；从学术的角度来讲，之前关于Page Coloring技术的工作都仅用来管理一个层次的存储资源，同时在整个内存层次结构上使用Page Coloring的优化方式还鲜有可参考的资料可查。

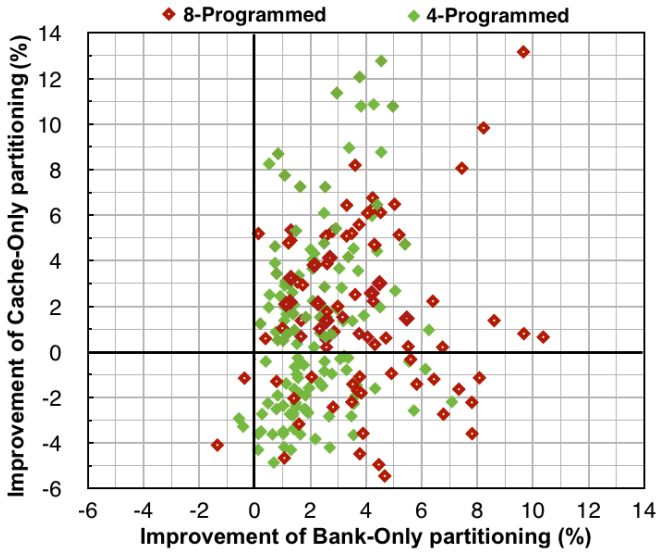

图 7.214 组工作集的性能分布（i7-860, 2.4GHz, 8GB DDR3）

最新的研究工作表明[9]，被广泛使用的多核系统的地址映射中，除了包含能索引Bank的地址位(Bank bits, B-bits)和索引Cache的地址位(Cache bits, C-bits)之外，还存在一类Bank与Cache重合的地址位，能够同时索引DRAM Bank和Cache set（在[9]中，被命名为O-bits，如图8所示）。从技术的角度来讲，如果对O-bits

进行“着色”，即按照重合的地址位进行划分，就可以同时划分DRAM Bank和LLC。[9]形象的将这种划分方式称之为**“垂直”划分（Vertical Partitioning，VP）**。在真实环境下的实验结果表明，如图9所示，对于在图7中第一象限的工作集，VP带来的性能提高要优于单个层次上的Bank和Cache划分。

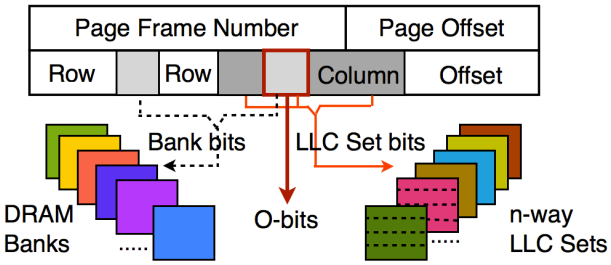

图 8. 地址映射中索引不同资源的地址位

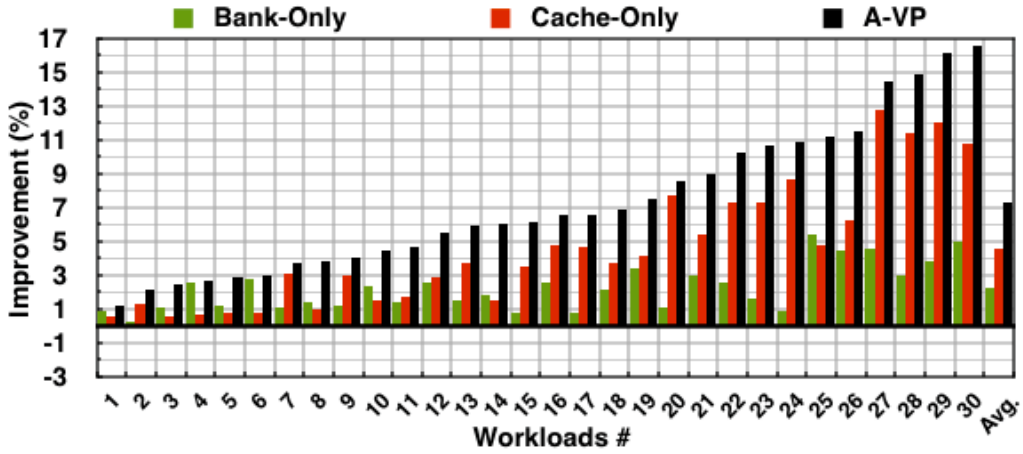

图9. 针对4-programmed工作集的A-VP性能效果图（i7-860, 2.4GHz, 8GB DDR3）

“垂直”划分能够更大程度的发挥出Page Coloring的技术潜力。[9]中的研究工作表明，普遍的情况是，如果工作集在Cache划分和Bank划分的情况下均有效（图7中第一象限中的工作集），那么VP将有可能带来更大的性能受益，VP实际上是将两种划分的效果非线性的累加在了一起。

总结与未来

本文总结了从1985年到2014年间，近30来Page Coloring技术的历史、发展和最新的研究成果。在有些参考资料中，Page Coloring常常被赋予更宽泛的“OS-based Page Placement”的含义[15]。

2010年之后，业内对Page Coloring的应用前景表现出些许的悲观态度。主要原因包括：1）一些主流的新片设计厂商采用XOR或者更为复杂HASH方式索引Cache和DRAM，这使得Page Coloring技术不能够被简单直接的应用于这些机器上；2）某些计算环境采用了Super Page技术，也给Page Coloring技术的应用带来困难。

但是，也有多学者认为这项技术将会有新的应用场景。例如，随着异构、异质存储的计算平台逐步走入人们的视野（如同时包含NVM和DRAM的服务器），很多学者意识到了Page Coloring，从操作系统控制数据分布的角度出发，有可能会被应用于异构平台的资源管理与分配等方面，进而高效的发挥出每种存储材质的优势，提高系统整体性能；此外，在“Datacenter as a Computer”的时代，Page Coloring技术还有可能为数据中心中面向服务质量的优化、虚拟机安全、云计算平台资源管理等方面提供解决方案的参考。

参考文献

[1] TAYLOR, G., DAVIES, P., ANDFARAWAY,D, M. The TLB slice—A low-cost high-speed address translation mechanism. In ISCA-1990.

[2] Richard E. Kessler and Mark D. Hill, “Page Placement Algorithms for Large Real-indexed Caches”, In ACM Transactions on Computer Systems, Nov. 1992.

[3] Theodore H. Romer, Dennis Lee, Brian N. Bershad and J. Bradley Chen, Dynamic Page Mapping Policies for Cache Conflict Resolution on Standard Hardware, In Proceedings of the First Symposium on Operating Systems Design and Implementation, Nov. 1994.

[4] E. Bugnion, J. M. Anderson, T. C. Mowry, M. Rosenblum, and M. S. Lam. Compiler-directed page coloring for multiprocessors. In ASPLOS-1996

[5] T. Sherwood, B. Calder, and J. Emer. Reducing cache misses using hardware and software page placement. In ICS-1999.

[6] S. Cho and L. Jin. Managing distributed, shared L2 caches through OS-level page allocation. In MICRO-2006

[7] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In HPCA-2008.

[8] Lei Liu, Zehan Cui, Mingjie Xing, Yungang Bao, Mingyu Chen, Chengyong Wu. A Software Memory Partition Approach for Eliminating Bank-level Interference in Multicore Systems. In PACT-2012

[9] Lei Liu, Yong Li, Zehan Cui, Yungang Bao, Mingyu Chen, Chengyong Wu. Going Vertical in Memory Management: Handling Multiplicity by Multi-Policy. In ISCA-2014

[10] Lei Liu, Zehan Cui, Yong Li, Yungang Bao, Mingyu Chen, Chengyong Wu. BPM/BPM+: Software-based Dynamic Memory Partitioning Mechanisms for Mitigating DRAM Bank-/Channel-level Interferences in Multicore Systems. In TACO-2014

[11] Wolfgang Petersen, An acknowledgement letter from Intel to the authors of [7] on adopting their cache partitioning software methods in production systems, 2010.

[12] Yoongu Kim, Michael Papamichael, Onur Mutlu, and Mor Harchol-Balter, Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior. In MICRO-2010.

[13] Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda. Reducing Memory Interference in Multicore System via Application-Aware Memory Channel Partitioning. In MICRO-2011.

[14] W. Mi, X. Feng, J. Xue, and Y. Jia. Software-hardware cooperative DRAM bank partitioning for chip multiprocessors. In NPC-2010.

[15] Rajeev Balasubramonian, Norman Jouppi, Naveen Muralimanohar. Multo-Core Cache Hierarchies. Copyright 2011 by Morgan & Claypool.

致谢

感谢前期与笔者一起讨论的学者，还要特别致谢美国俄亥俄州立大学张晓东教授对笔者科研工作的关心和支持。对本文的校审人员一并致谢。