**体系结构组-论文阅读报告**

阅读人：姚文涛

论文题目：SAC:Sharing-Aware Caching in Multi-Chip GPUs

论文出处：ISCA

发表时间：2023

论文作者（仅列通讯作者）：Lieven Eeckhout

作者单位：Ghent University

论文摘要：

多GPU系统中，为了克服片间和片内带宽的不同，设计出性能最优的LLC是一件十分困难的挑战。在先前的工作中，内存端LLC可以最大化内存利用率，但是在访问远端数据时，不得不经历带宽小的片间网络；SM端LLC可以缓存远端数据，但是其对内存的利用率比较低，还得保持缓存一致性以增加设计难度。同时，作者发现，不同的workload在这两种LLC配置下有着不同的偏好，并且LLC之前的有效带宽对系统性能有着至关重要的影响。由此，作者团队想出了一种sharing-aware caching的策略来对上述两种LLC配置进行选择，以使得LLC配置能够满足不同workload的偏好，从而达到性能的全局最优解。该策略由片内网络和LLC控制器动态地配置最大化workload性能的LLC组织方式。同样，SAC也通过通俗易懂的EAB模型来预测两种LLC组织方式对其有效带宽带来的影响，从而控制LLC controller进行配置。与先前的工作相比，SAC平均性能上获得了一些提升，同时也跳出了局部最优解的尴尬。该团队展示了在跨设计空间和跨workload上有着巨大的性能进步。

（1）论文背景：

随着对计算和内存密集访问应用的需要，具有高并行度的GPU成为十分重要的加速器，通常情况下，我们通过增加SM数量和拓展内存带宽来对GPU的性能进行提升。其代价就算面积的增加和产量的降低。增加GPU的规模带来的提升往往会比增加单GPU性能带来的提升更大。

多GPU的架构通常分为multi-socket GPUs和MCM GPUs两类。前者把很多GPU和内存放到同一块PCB上，并通过内部互联总信念进行通讯，具有成本低，内存容量高，带宽低的特性；后者将GPU和内存分别封装具有高带宽、内存容量低、带宽高的特性。目前对于多GPU架构，主要的挑战就是去克服片间和片内带宽不同所带来的影响。

由于传统观念认为，GPU性能的影响是由于内存带宽引起的。先前的工作中，人们大多是将重点放在了内存带宽的优化上。最近，人们意识到LLC带宽对GPU性能的影响，开始着手在LLC上进行设计和改进。一般来说，GPU的LLC分为内存端和SM端，前者可以最大化内存的利用率，但是如果访问的数据过于遥远，必须得经历带宽小的链路；后者可以缓存远端的数据从而使得访存数据的时候带宽加大，缺点就是内存利用率比较低，需要保持缓存一致。两个比较经典的例子就是Milic等人提出的动态LLC策略和Arunkmar等人提出的1.5机缓存的概念。

（2）论文解决的问题和提出的思路：

一般情况下有两种形式的LLC组织方式。Memory-Side(内存端)和SM-Side(SM端)。前者cache data中存储着本地数据的数据，后者的cache data存储着整个内存数据。后者的NoC因为不需要与SM和LLC之间的通信抢占资源，所以其能提供固定延迟和带宽优势，但是因为要维持一致性协议，会导致更大的面积和功耗。

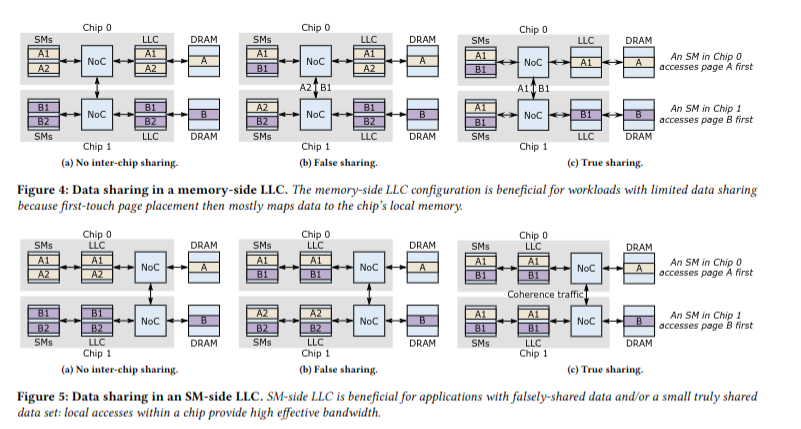


Figure 1: 共享方式

我们现在讨论cache line的三种情况：不共享，伪共享，真共享。在不共享的情况下两种LLC组织形式没什么差别。在伪共享的情况下，内存端LLC的组织形式会导致访问远端数据时，重复访问所遍历过的芯片链路，SM端LLC的组织形式则避免了链路的重复遍历。这么看SM端更优，因为其将经常访问的数据缓存在LLC上来增大有效带宽。在真共享的情况下，内存端LLC组织形式会增加芯片间链路负担，但其避免一致性的问题，并产生更高的LLC利用率，但他并不会对共享缓存展示出很大的有效带宽，而SM端增加了芯片间的有效带宽，但是减小了LLC容量，同时也需要一致性协议支持。SM端LLC在两种情况下有优势：

1. 复制真缓存行所带来的带宽优势 比 抵消LLC未命中率所增加的带宽开销 大。
2. 有足够多的共享数据来抵消维持一致性协议的开销。

本文先假设了一个4 chips GPU去进行分析，并进行了实验，如下图所示：

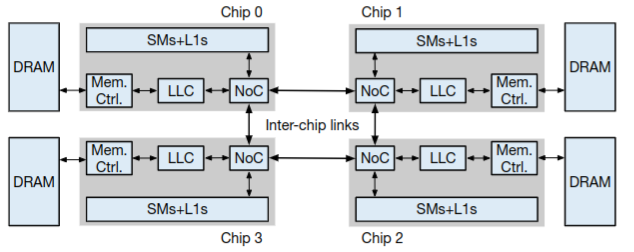


Figure 2: baseline multi-chip GPU的方案

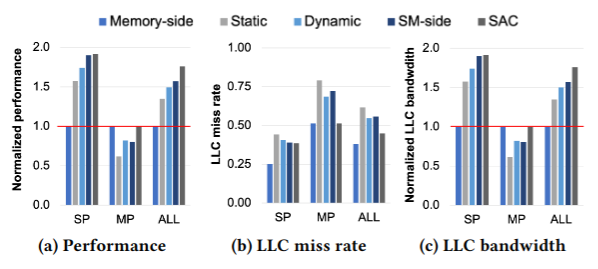


Figure 3: 不同LLC组织下的4-chip GPU在SM-Side preferred(SP) and memory-side preferred(MP)benmarks的性能表现，LLC未命中率，LLC带宽

结果发现在SP的时候，SM端的LLC miss rate要比内存端要大，但平均性能却成反比关系（体现在平均性能提高了91%）。这十分的反直觉，但这种现象往往发生在带宽增加能够削减高额的未命中率的情况下，原因是SM在访问远端数据时，数据经过本地cache的高带宽的芯片内网络，而并不是经过芯片间的低带宽网络。同时，这样链路延迟的缩小所带来的性能提升，比高miss rate带来的性能削减大。所以，平均性能反倒提升了不少。但是对于MP，带宽的增加不能削减高miss rate所带来的影响，所以，内存端LLC组织方式比SM端平均性能提高32%。

传统观念认为，GPU的性能影响是由于内存带宽引起的，但根据上述分析，可以看到LLC带宽对GPU性能的影响。事实上，workload通常会根据哪种类型的LLC带宽更大去选择LLC的组织形式，而这取决于workload中芯片间数据共享的程度和性质。对于小的数据集，会偏向SM端的LLC组织形式，因为这样会有足够多的LLC空间去复制其共享数据。对于大的数据集则会更偏向内存端的LLC组织形式，因为LLC的空间不足以存放这么大的数据集。无论是之前提出的动态的还是静态的LLC都无法最大化LLC的有效带宽。本文则提出了在内存端LLC进行微小改动的Sharing-Aware Caching（SAC）来对SM端和内存端的LLC组织形式进行选择，以最大化LLC的有效带宽

（3）论文提出的方法：

SAC作为一种LLC组织形式，在Memory-Side LLC的基础上进行了微小的改动，主要增加了两个部分:旁路，CRD，并提出EAB模型用来分析最优的LLC组织形式。其初始化为内存端LLC，刚开始性能计数器负责收集数据，送往分析窗口，分析窗口用EAB模型计算出2k个时钟周期内内存端和SM端LLC组织形式的结果并进行比较，当该芯片SM端结果要比内存端的EAB要高出大约5%的时候，采用SM端；反之，则采用内存端LLC组织形式。

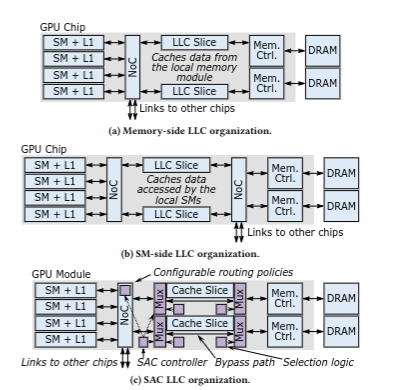


Figure 4: LLC三种组织形式

SAC的LLC组织形式如图4(c)所示，在内存端LLC组织形式的基础上增加了一个逻辑选择电路用来决定是否绕过LLC,这个旁路(也就是图中的双箭头)连接着内存控制器和一个在内存端和SM端LLC上的多路复用器/解复用器。如果当前配置为SM端LLC，则请求会被复用到LLC或者旁路上；响应则会被复用到NoC端口。如果当前配置为内存端LLC，则响应就会解复用到SM和NoC端口，请求会被复用。同时，本地未命中和远端未命中在这里共享一个请求队列。如果队列满了，就会强行让请求在本地LLC前等待。其优点不言而喻，它不需要对crossbar结构进行修改，只是在外围增加了两个逻辑选择，一是将SM未命中从本地转发到NoC，二是当远端SM未命中的时候，会经过LLC

下图则是两种LLC配置下未命中的时候经过NoC路径说明：

对于SL和ML类型的未命中，二者的路径是一样的，都是直接访问本地的内存。对于SR类型的未命中，请求先到达LLC，然后再到达NoC进行路由，接着如现在远端芯片上，并通过旁路访问内存，响应沿着原路返回。对于MR类型的未命中，请求被回直接送往NoC进行路由，然后送到远端芯片上并依次对LLC和内存进行访问。

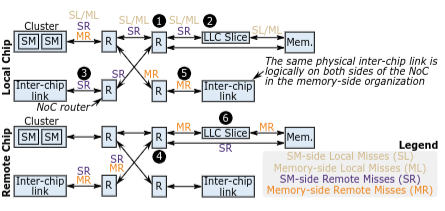


Figure 5: 内存端和SM端LLC未命中的路径

对于分析窗口大小，最开始的2k个时钟周期十分恰当，这能够保证Chip Request Directory(CRD)看到数据地址映射到此内存分区的所有请求。而且只需要在最开始运行一次，之后无需再次运行（再次运行给性能的提升很小，而且会带来额外的负担）。

本文还提出了Effective Available Bandwidth(EAB)模型作为分析LLC配置最优化的依据。事实上有效带宽是在给定工作负载的情况下，对某个模块进行访问，系统所能提供的最大带宽。最原始的公式为：

这里两个加数分别为本地和远端系统能够提供的有效带宽，具体计算公式如下：

远端或者本地的EAB值，是SM和LLC之间的带宽，以及LLC命中所呈现出的有效带宽和 LLC未命中的有效带宽、LLC和内存之间的带宽和内存的带宽的最小值 的之和的最小值。

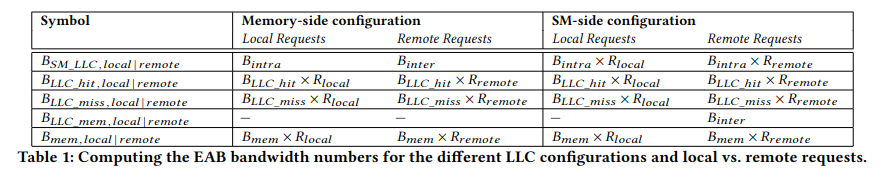
其计算主要取决于两点，一是他们是否关注本地或者远端请求，二是设特定的LLC组织形式

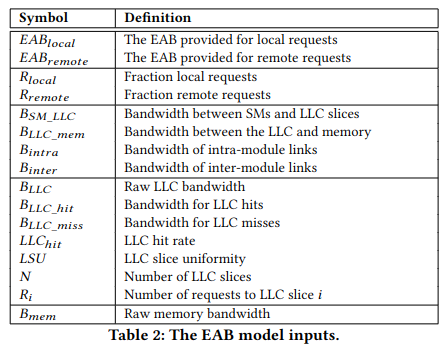
下面对前面提到的集中带宽进行详细叙述：

对于SM和LLC之间的有效带宽，为GPU内部网络带宽乘以相应（本地/远端）的请求比例系数。LLC命中的有效带宽为 LLC原始带宽×LLC的一致性（LSU）×LLC命中率，相反，LLC未命中的有效带宽为LLC原始带宽×LSU×LLC未命中率。其中LSU的公式如下：

在这个公式中，为第i个LLC的请求个数。

下面为详细的参数表：





从以上模型所需参数可以看到有些参数是只由架构决定的，比如,和，还有一些参数只有workload决定，也有一些参数由着二者共同决定。

基于此，我们提出了一种用来统计EAB模型参数的计数器，和在内存端LLC配置情况下能够预测SM端配置下的未命中Chip Request Director(CRD)。如下图所示：

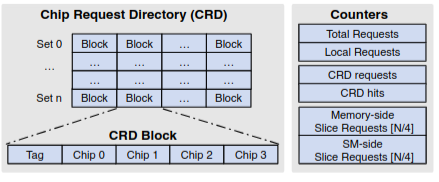
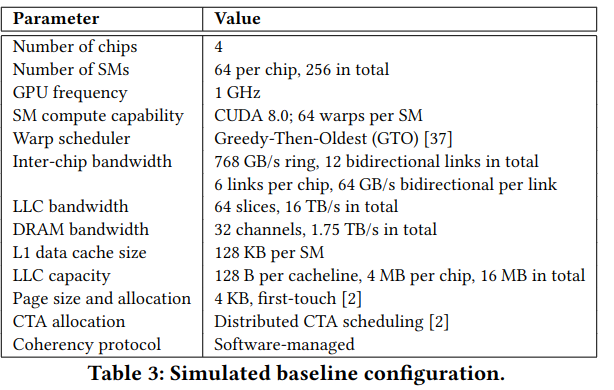


Figure 6：CRD和计数器

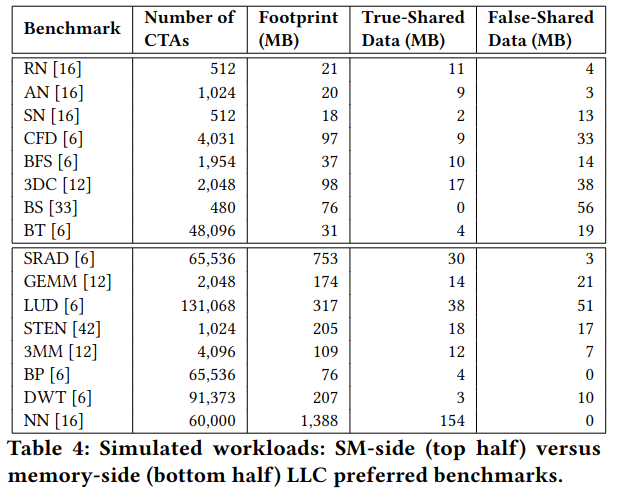
对于计数器，Total requests,local requests用来计算,slice request用来计算LSU，CRD hits counters, CRD requests counters用来计算SM端LLC配置下的未命中率。CRD采样n组本地缓存区，去记录在SM端LLC配置下的任何芯片中该块是否被访问和复制。这样，每一个块都由一个tag位和4位标志位组成，标志位用来记录是否被chip i访问到。如果有分组，还要增加位去看到底是哪组被访问到了

1. 论文开展的实验：

该论文首先在GPGPU-Sim上模拟4芯片的多GPU系统，采用DSENT进行NoC仿真，CACTI进行内存/缓存的仿真。



具体参数如上表所示，每个芯片上64个SM分为32个SM簇，即每两个SM共享一个网络节点，NoC配置为hierarchical crossbars并且考虑环形拓扑结构，以模拟集中式的crossbar，这种crossbar往往会有着更大的面积和功耗，芯片中间的连接采用第二代NVLink协议，内存接口采用GDDR6。同时，每个SM的L1cache配置为写通，LLC为写回，并且采用软件形式的缓存一致性协议。并且采用4个内存分区和32个内存通道。测试所采用的workload如下表所示



在测试中，我们进行五种LLC配置，分别为：Memory-Side LLC, SM-Side LLC, Static LLC, Dynamic LLC, SAC

在实验中，作者根据跑分程序对LLC组织形式的偏好进行了分组，左边是偏好SM端LLC组织形式，右边则是偏好内存端的LLC组织形式。最后，我们对两种类型的跑分程序分别进行了平均。如图7所示，SAC的LLC组织形式是效果最好的。也不难发现，SAC在SP下的结果和SM端LLC很相似，同样，在MP下的结果和内存端LLC几乎一样，这是由于SAC能在分析窗口结束后，去配置LLC的组织形式导致的，这也表示，EAB模型是能够成功的分析出跑分程序的偏好并指导旁路进行逻辑选择的。

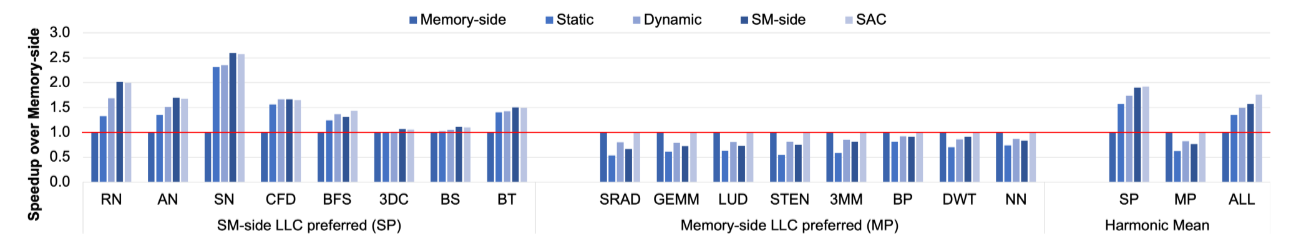


Figure 7: 不同LLC配置相对于Memory-Side LLC的加速

为了搞清楚为什么SAC比其他LLC组织形式更优，图8展示了不同LLC组织形式下本地和远端数据缓存占比。Static LLC无论在何种跑分程序下本地和远端数据缓存占比为1:1。对于SP，Dynamic LLC会比Static LLC缓存更多的远端数据，但是SM-Side LLC和SAC能够缓存最多的远端数据，这能够显著提高缓存的有效带宽。对于MP， 虽然SM-Side LLC和Dynamic LLC能够缓存更多的本地数据，但是SAC只缓存本地数据，并不会缓存任何远端数据。

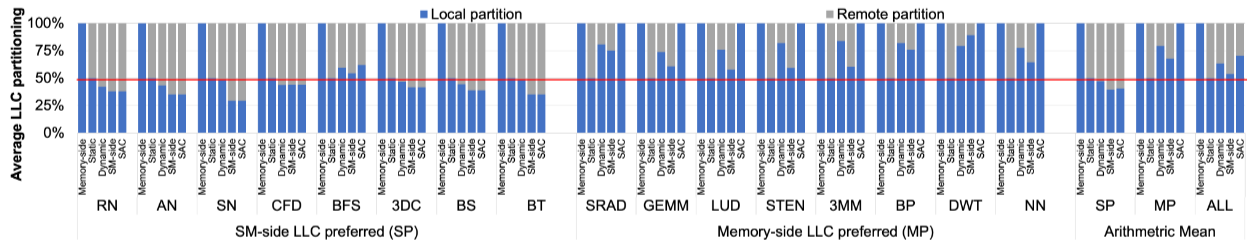


Figure 8: 量化不同LLC组织形式是如何缓存本地和远端数据

SAC选择SM-Side LLC还是Memory-Side LLC从根本上说是由workload决定的，想当然的认为部分缓存本地数据，部分缓存远端数据是不可行的。内存端LLC和SM端LLC的选择归根到底还是跑分程序是MP还是SP。由此观之，Static LLC太笨重了，它在SP的时候不能够缓存足够多的远端数据，在MP的时候也不能缓存很多的本地数据。Dynamic LLC虽然看起来比Static LLC聪明一些，但其启发式算法，也往往会陷入局部最优解的困局中。SAC解决了这个局部最优解的问题，达到了全局最优解。

通过SAC获得的性能提升与LLC的有效带宽强相关，结果如图9所示。该图中对LLC有效带宽的提升进行分类，即分为本地LLC，远端LLC，本地内存和远端内存。对于SP，由于片内网络带宽比片间网络带宽更高，SAC往往会选择将对远端LLC的访问替换成本地LLC的访问从而对有效带宽进行提升。

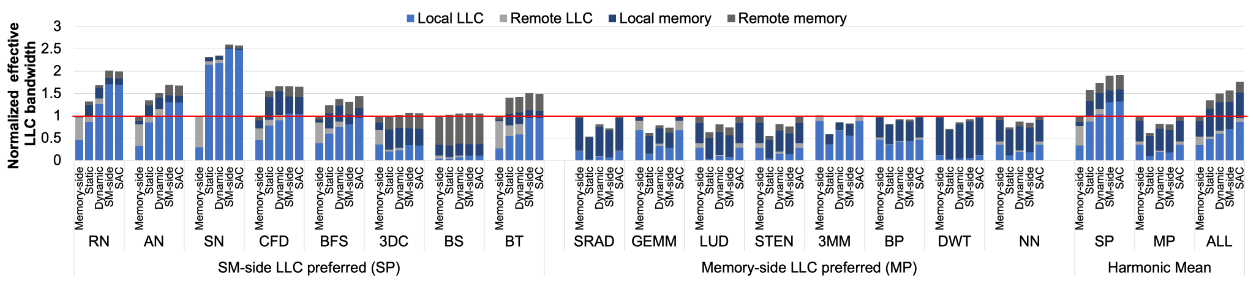


Figure 9: 标准化LLC有效带宽并展示每个周期LLC响应的来源及其数量

为了进一步探究有效带宽取得提升的原因，作者改变工作集的和分析窗口大小，来探究其影响。由图10可以得出如下结论。首先，turly sharing working set在SP下是相对较小的。Turely sharing working set一般在SM的配置下不同芯片访问同一个cache line的时候进行复制，而跨芯片复制相对较少的turely sharing working set并不会对系统造成很大压力，但其能够显著增加有效带宽。相反， MP有相当大的truely sharing working set, 复制这些working set往往超过了LLC的容量，从而造成cache trashing，故在此情况下，这些数据集会更喜欢内存端的配置。

其次，对于false-sharing working set在SP下相当大，SM配置下的有效带宽要大于内存端配置下的有效带宽。

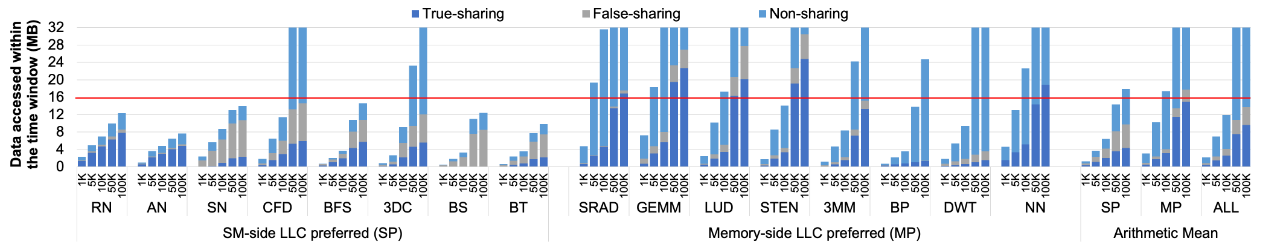


Figure 10: 在SM-Side LLC配置下，不同窗口长度的数据集大小

在上文所叙述的实验中，有一个例外是BFS，其SAC要远优于SM端LLC，图11说明了BFS执行的时候每个内核的配置选择。

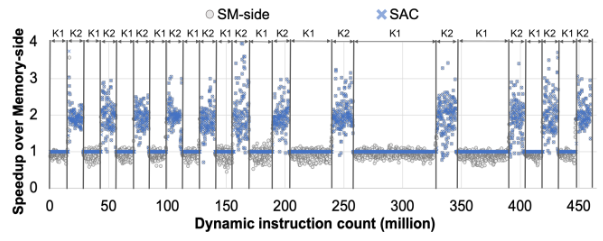


Figure 11: BFS的时变行为

由于SAC的优秀表现来自总working set相关的shard working set，所以很有必要去评估输入大小的灵敏度。图12,我们可以得到，SAC会根据输入的大小去选择最优的LLC配置。比如说，对于SP，SAC往往会选择SM端LLC进行配置，但是当输入规模比较大的时候，因为太大的shared working set会导致cache trashing，就会转变进而选择内存端LLC进行配置。反之，对于MP，则会在相反的情况下进行相反的配置

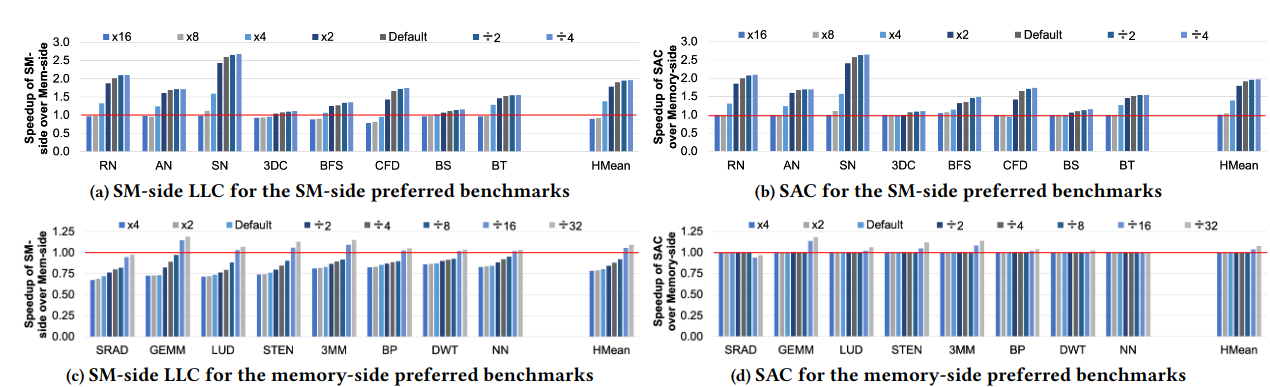


Figure 12: SP的输入数据集的灵敏度分析

由图13可以得出，SAC的在内存端的性能增加会随着片间带宽增加而减少，主要原因是片间带宽越高，本地缓存远端数据和复制共享数据的重要性就越低。由于更大的LLC可以在本地缓存更多的数据，并且能够跨芯片复制更大的shared working set，则与内存端LLC相比，更大的LLC容量提高了SAC的性能。从当内存接口从GDDR5改进到GDDR6和HBM2的时候，系统瓶颈从内存带宽转移到片间网络上。因此复制共享数据集对于高带宽内存系统更为重要，它可以提高SAC的性能优势。从一致性的角度说，硬件一致性的效果要比软件一致性要好。同时，SAC的性能会随着GPU个数的增加而增加。还要额外说明的是，SAC有效带宽对page size并不是特别敏感，因为其只影响了false sharing, 而ture sharing才是提升SAC有效带宽的关键。

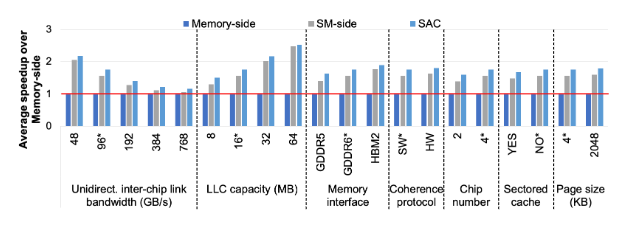


Figure 13:灵敏度分析

（5）论文结论：

本文展示了一种sharing-aware caching的策略来通过对刚开始的workload对内存访问的数据进行分析，来动态地配置每一个芯片的LLC组织方式是内存端还是SM端。其主要优势在于：1.对内存端LLC进行微小改动即可变为SM端配置，以及其简单的判断逻辑电路也带来了设计上的便捷 2.使用简单且朴素的EAB模型能够对两种LLC组织形式的有效带宽进行分析，从而判断出更优的LLC组织形式。其相比于在SM端和内存端LLC基础上进行改进的Static LLC以及Dynamic LLC，能够跳出局部最优解的困局，达到全局最优解的最终目的。

本文还告诉我们并不是某一种LLC架构是最优的，在特定的workload下采用特定的LLC组织形式才能达到最优的性能。而这并不是一成不变的，当输入的量过大的时候，SM端的配置也会被迫重新选择内存端的配置；反之，当输入的量过小的同时，内存端的配置也会重新选择SM端的配置。而且由于系统瓶颈从内存带宽转移到片间网络上，复制共享数据集对提升性能显得尤为重要。同时，LLC有效带宽主要被ture sharing的数据影响，其对false sharing并不是特别敏感。最大化的利用ture sharing的复制也就正是SAC的最大优势。

同时， GPU的数量越多，SAC的性能也就越高。同时，增大LLC容量也是有用的，毕竟，其能够在本地缓存更多的数据，并复制更多的sharing working set, 增加的带宽能够削减高额未命中率带来的负面影响

（6）心得和收获

这篇文章作为我在体系结构方面阅读的第一篇论文，初看时，感觉其思路惊为天人，感叹作者的才高八斗。细看此文时，感觉一切又是那么的朴素，但是又是那么智慧，全文无处不透露这作者扎实的知识体系，和严谨的科学素养。

通过对整篇文章的研读，对于内存、GPU和NoC的基础知识学会不少。

从整体思路上来看，本文先介绍了背景并提出了设计LLC最优架构的问题，然后由Memory-Side LLC和SM-Side LLC架构引出SAC的架构，然后具体分析了上述LLC组织形式对于不同数据共享形式的优劣。接着，从旁路、运行时所需的支持、EAB模型三个角度对SAC策略进行了详细的阐述。最后，对其进行步步深入的实验，从多个角度，全方位的探究了为什么SAC策略能够达到最优策略。整体思路十分清晰，从提出问题，到分析问题，到解决问题，最后再对解决方案进行进一步思考与探讨，也是对科研思路的一个很好的总结。短短12页的正文，工作量十足。

从解决方案上看，本文的解决思路十分有趣。因为传统的思想都是在原有的某一个架构基础上进行更改，从而达到更优，但是特定架构和通用架构恰巧是两个很矛盾的成分在里面，当我们要在某一个特定的方面达到最优的时候会采取针对该方面的特殊优化，而通用架构则会为了保持在更广的情况跑出一个还不错的成绩，往往会放弃特定的高效性。本文提出的思路的角度是通过将两个原本存在的策略揉在了一起，使其进行优势互补。从而达到最优解，这就又有点太极中的阴阳鱼的意思在里面了。这也在提醒着，在做通用架构的优化是，可以参考特定架构的优化方案，找到能够完美互补的两种甚至多种策略，将其巧妙的揉在一起，从而提升性能。

从验证方法上看，本文中验证时，对系统进行了详细的定义与明确，采用很多跑分程序，以使得结论本身的偶然性并不是那么的强。其也是由浅入深，再进行广度的分析。从探究不同LLC架构对workload性能的影响，这一最浅显的关系入手，紧接着，通过两次实验深入探究了LLC架构在运行不同workload的数据来源。随后将广度铺开，显示探究了分析窗口和输入量大小以及时变对其影响，紧接着将广度进一步铺开，从片间带宽，LLC容量，内存接口，一致性协议，GPU数量和page size这几个角度进行探究。以后分析系统时，感觉也可以先通过这个几个固定的角度去分析一下。验证时，广度和深度兼具。先将问题探究其最本质，然后在通过影响最本质的几个参数，去对问题进行广度上的探究。这样先深度，后广度的思考本质的方式也十分值得我去学习。尤其是当我深度思考问题的时候不要忘记对问题进行广度上的探索，要不然很容易落得“不识庐山真面目，只缘身在此山中”局面。