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。

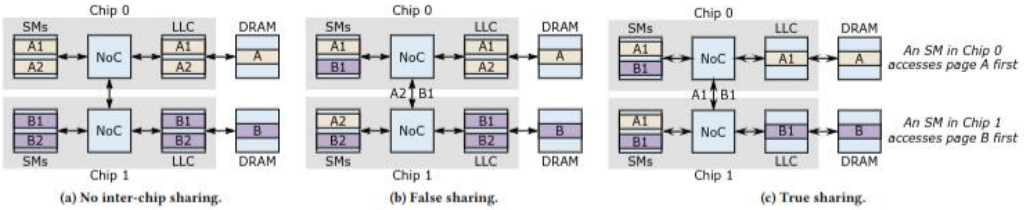

Figure 4: Data sharing in a memory-side LLC. *The memory-side LLC configuration is beneficial for workloads with limited data sharing because first-touch page placement then mostly maps data to the chip's local memory.*

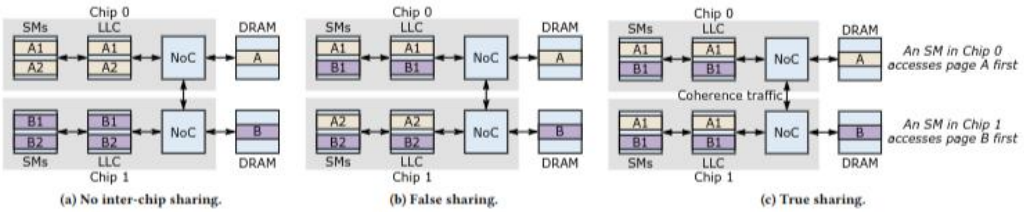

Figure 5: Data sharing in an SM-side LLC. *SM-side LLC is beneficial for applications with falsely-shared data and/or a small truly shared data set: local accesses within a chip provide high effective bandwidth.*

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：

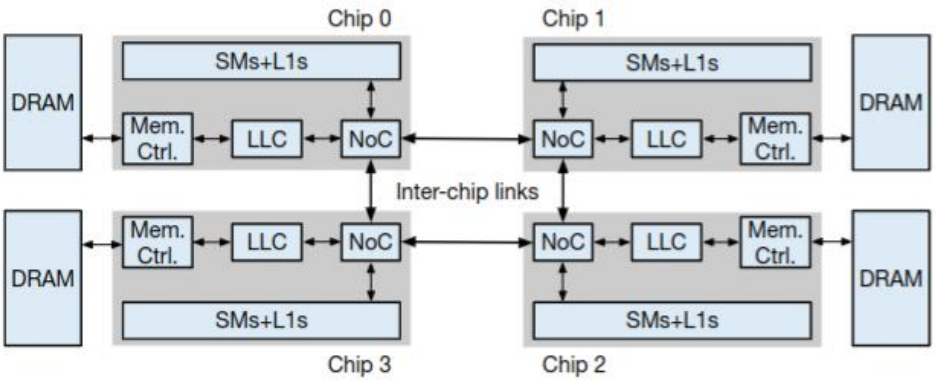

Figure 2: baseline multi-chip GPU 的方案

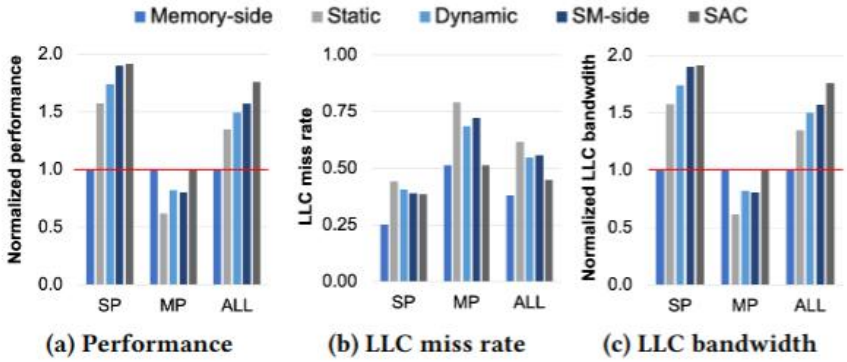

Figure 3: 不同 LLC 组织下的 4-chip GPU 在 SM-Side preferred(SP) and memory-side preferred(MP)benmarks 的性能表现，LLC 未命中率，LLC 带宽

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。

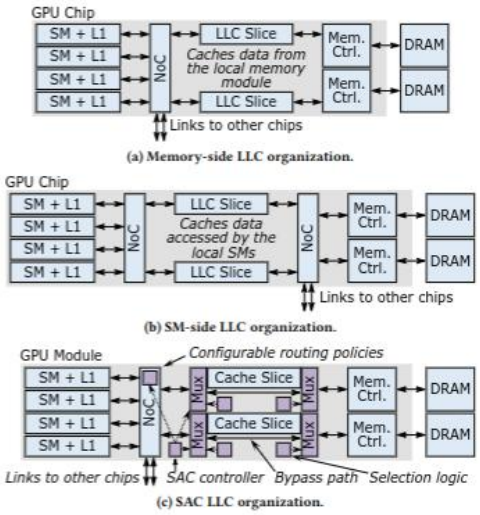

Figure 4: LLC 三种组织形式

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。

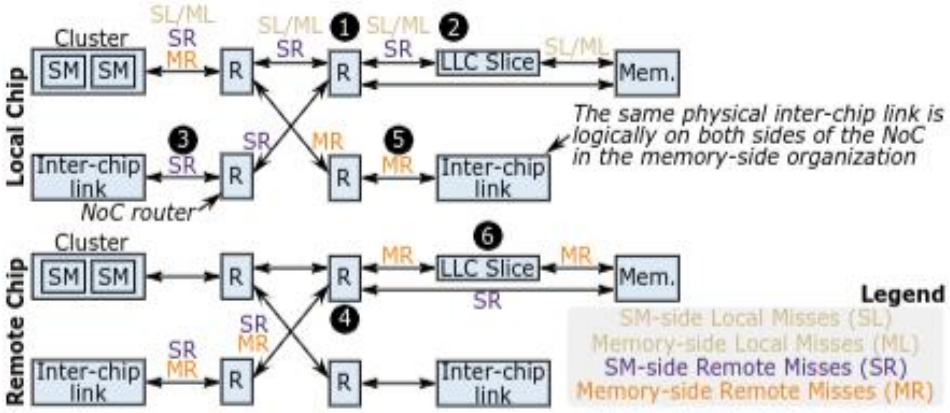

Figure 5: 内存端和 SM 端 LLC 未命中的路径

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：

$$EAB_{total} = EAB_{local} + EAB_{remote}$$

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：

$$EAB_{local\ |\ remote} = \min(B_{SM\_LLC}, B_{LLC\_hit} + \min(B_{LLC\_miss}, B_{LLC\_mem}, B_{mem}))$$

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：

$$\mathrm{LSU} = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \frac{\mathrm{R_i}}{\max(\mathrm{R_1}, \ldots, \mathrm{R_N})}$$

在这个公式中，$\mathrm{R_i}$为第 i 个 LLC 的请求个数。

下面为详细的参数表：

Symbol	Memory-side configuration		SM-side configuration	
	Local Requests	*Remote Requests*	*Local Requests*	*Remote Requests*
$B_{SM\_LLC,local\|remote}$	B_{intra}	B_{inter}	$B_{intra} \times R_{local}$	$B_{intra} \times R_{remote}$
$B_{LLC\_hit,local\|remote}$	$B_{LLC\_hit} \times R_{local}$	$B_{LLC\_hit} \times R_{remote}$	$B_{LLC\_hit} \times R_{local}$	$B_{LLC\_hit} \times R_{remote}$
$B_{LLC\_miss,local\|remote}$	$B_{LLC\_miss} \times R_{local}$	$B_{LLC\_miss} \times R_{remote}$	$B_{LLC\_miss} \times R_{local}$	$B_{LLC\_miss} \times R_{remote}$
$B_{LLC\_mem,local\|remote}$	–	–	–	B_{inter}
$B_{mem,local\|remote}$	$B_{mem} \times R_{local}$	$B_{mem} \times R_{remote}$	$B_{mem} \times R_{local}$	$B_{mem} \times R_{remote}$

Table 1: Computing the EAB bandwidth numbers for the different LLC configurations and local vs. remote requests.

Symbol	Definition
EAB_{local}	The EAB provided for local requests
EAB_{remote}	The EAB provided for remote requests
R_{local}	Fraction local requests
R_{remote}	Fraction remote requests
B_{SM_LLC}	Bandwidth between SMs and LLC slices
B_{LLC_mem}	Bandwidth between the LLC and memory
B_{intra}	Bandwidth of intra-module links
B_{inter}	Bandwidth of inter-module links
B_{LLC}	Raw LLC bandwidth
B_{LLC_hit}	Bandwidth for LLC hits
B_{LLC_miss}	Bandwidth for LLC misses
LLC_{hit}	LLC hit rate
LSU	LLC slice uniformity
N	Number of LLC slices
R_i	Number of requests to LLC slice i
B_{mem}	Raw memory bandwidth

Table 2: The EAB model inputs.

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,$\mathrm{B_{intra}}$和$\mathrm{B_{inter}}$，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：

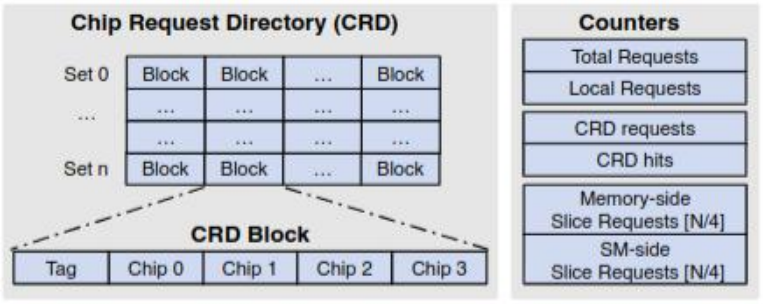

Figure 6：CRD 和计数器

对于计数器，Total requests,local requests 用来计算 R_{local} ,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

（4）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。

Parameter	Value
Number of chips	4
Number of SMs	64 per chip, 256 in total
GPU frequency	1 GHz
SM compute capability	CUDA 8.0; 64 warps per SM
Warp scheduler	Greedy-Then-Oldest (GTO) [37]
Inter-chip bandwidth	768 GB/s ring, 12 bidirectional links in total 6 links per chip, 64 GB/s bidirectional per link
LLC bandwidth	64 slices, 16 TB/s in total
DRAM bandwidth	32 channels, 1.75 TB/s in total
L1 data cache size	128 KB per SM
LLC capacity	128 B per cacheline, 4 MB per chip, 16 MB in total
Page size and allocation	4 KB, first-touch [2]
CTA allocation	Distributed CTA scheduling [2]
Coherency protocol	Software-managed

Table 3: Simulated baseline configuration.

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

Benchmark	Number of CTAs	Footprint (MB)	True-Shared Data (MB)	False-Shared Data (MB)
RN [16]	512	21	11	4
AN [16]	1,024	20	9	3
SN [16]	512	18	2	13
CFD [6]	4,031	97	9	33
BFS [6]	1,954	37	10	14
3DC [12]	2,048	98	17	38
BS [33]	480	76	0	56
BT [6]	48,096	31	4	19
SRAD [6]	65,536	753	30	3
GEMM [12]	2,048	174	14	21
LUD [6]	131,068	317	38	51
STEN [42]	1,024	205	18	17
3MM [12]	4,096	109	12	7
BP [6]	65,536	76	4	0
DWT [6]	91,373	207	3	10
NN [16]	60,000	1,388	154	0

Table 4: Simulated workloads: SM-side (top half) versus memory-side (bottom half) LLC preferred benchmarks.

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。

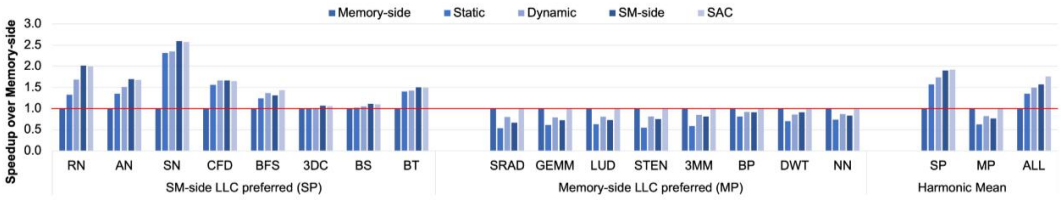

Figure 7: 不同 LLC 配置相对于 Memory-Side LLC 的加速

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。

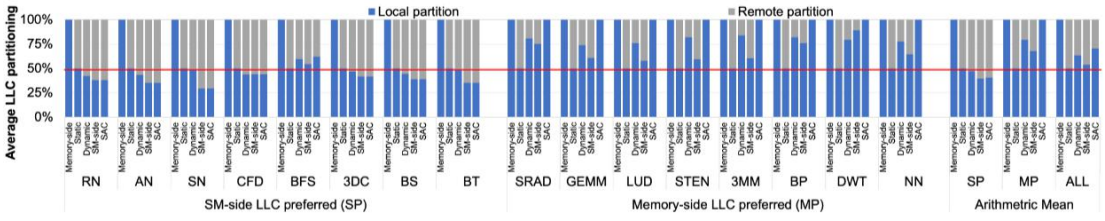

Figure 8: 量化不同 LLC 组织形式是如何缓存本地和远端数据

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。

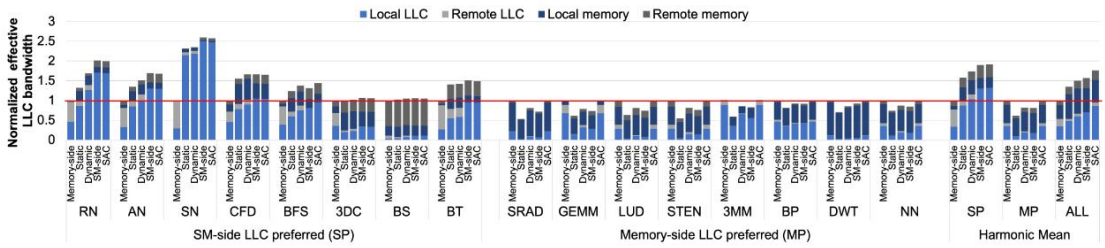

Figure 9: 标准化 LLC 有效带宽并展示每个周期 LLC 响应的来源及其数量

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。

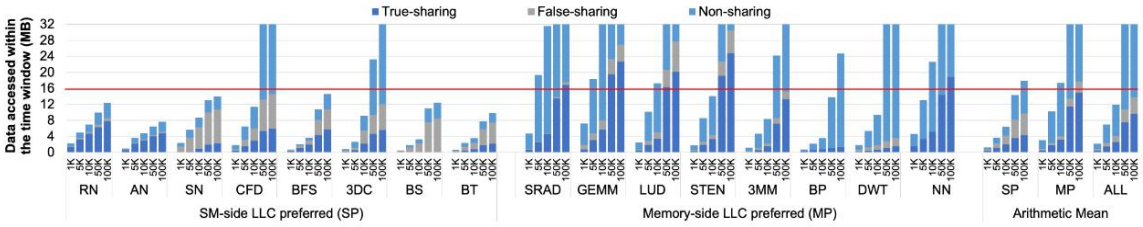

Figure 10: 在 SM-Side LLC 配置下，不同窗口长度的数据集大小

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。

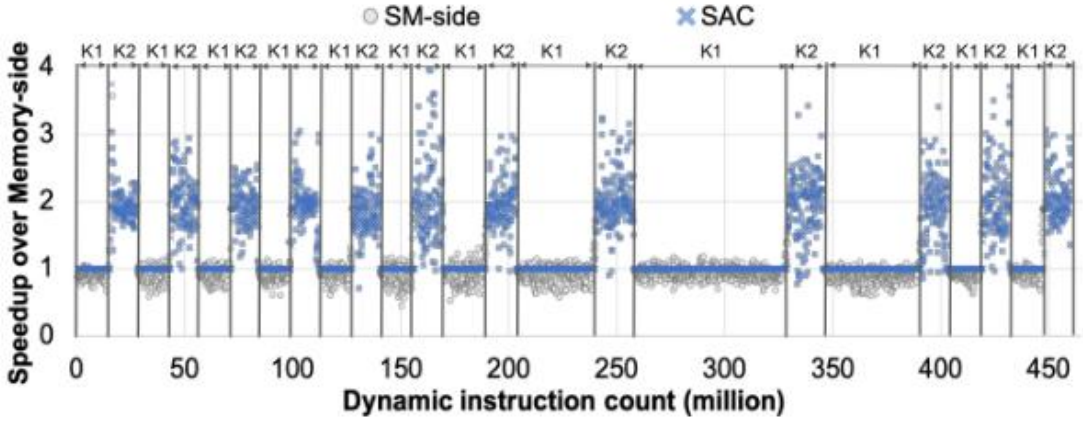

Figure 11: BFS 的时变行为

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

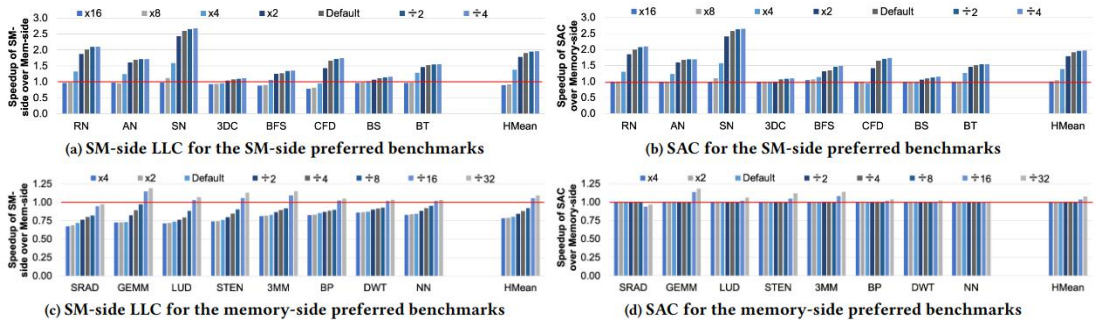

Figure 12: SP 的输入数据集的灵敏度分析

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。

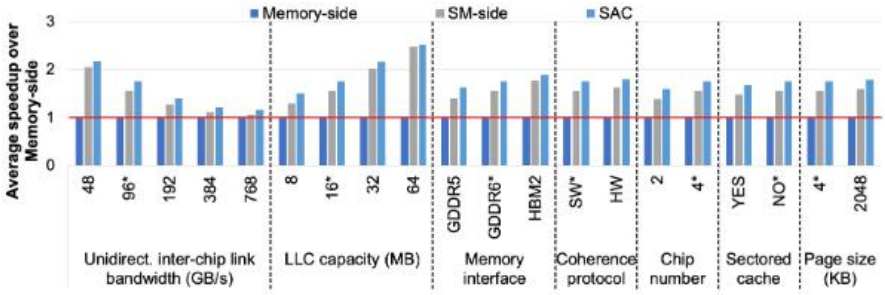

Figure 13:灵敏度分析

（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。