# SMILE: LLC-based <u>S</u>hared <u>M</u>emory Expansion to <u>I</u>mprove GPU Thread <u>L</u>evel Parall<u>e</u>lism

Tianyu Guo, Xuanteng Huang, Kan Wu, Xianwei Zhang[#], Nong Xiao
Sun Yat-sen University, Guangzhou, China
{guoty9,huangxt57,wukan3}@mail2.sysu.edu.cn,{zhangxw79,xiaon6}@mail.sysu.edu.cn

## ABSTRACT

While designed for massive parallelism, GPUs are frequently suffering from low thread occupancy and limited data throughput, which are typically attributed to constrained on-chip resources, such as shared memory and register file. To alleviate the pressure, last-level cache (LLC) is being substantially enlarged to support continuously growing computation and to shrink the off-chip data traffic. Nevertheless, applications can be challenging to fully utilize the excessive LLC spaces. Towards the issue, we propose to manage partial LLC in an software way instead to expand precious shared memory (SMEM), named as `SMILE`, helping to alleviate the low thread occupancy. `SMILE` splits the monolithic LLC into normal data cache and new software region, with the latter being to extend the limited SMEM. For adapting to diverse application characteristics, `SMILE` enables multiple splitting grades and determines the appropriate partition via online profiling. Experimental results show that `SMILE` achieves average performance improvements of 14.7% and 8.4% respectively, compared to the default baseline and prior state-of-the-art.

**ACM Reference Format:**

Tianyu Guo, Xuanteng Huang, Kan Wu, Xianwei Zhang[#], Nong Xiao. 2024. SMILE: LLC-based <u>S</u>hared <u>M</u>emory Expansion to <u>I</u>mprove GPU Thread <u>L</u>evel Parall<u>e</u>lism. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655906

## 1 INTRODUCTION

Designed for parallel processing, graphics processing units (GPUs) are adopted in a wide range of realms, including artificial intelligence, embedded systems and scientific computing. Driven by the ever growing computing demands, GPUs have evolved to feature intensive thread-level parallelism (TLP) and ample memory resources. Particularly, recent years witnessed the continuously increasing computation capabilities, and the growing memory capacity and bandwidth. For instance, Nvidia GPUs report an 8x surge on compute units (i.e., streaming multiprocessor, or SM) count, and meanwhile ceaselessly expand L1 data cache (L1D) and shared memory (SMEM) spaces, e.g., L1D/SMEM capacity has inflated from 64KB on Kepler to 256KB on Hopper.

Whereas with rising aggregated resources (e.g., larger L1), GPU compute units are suffering from degraded resource quota, which inevitably causes insufficient threads to be concurrently executed. From Kepler to Hopper, computing capability (TFLOPS) grows by 13.3x, while L1 size just poses a moderate 4x increase, conveying the exacerbated on-chip resource contention. To alleviate resource bottlenecks, a plethora of researches have been conducted to extend cache space [1–3], and optimize register file management [4, 5]. Besides, task mapping and scheduling [6, 7] are employed to reduce resource idling and speed up executions. To maximize TLP, techniques like software pipelining [8] have also been proposed to further expose inherent parallelism.

In addition to the private memory resources (e.g., registers and L1 cache) and execution units, SMs are sharing the last-level cache (LLC) across the entire GPU. Despite being considered to be critical to lessen the off-chip memory burden, conventional GPU LLC is of very limited size, which is thus highly competed among enormous threads [9]. To effectively utilize LLC, a couple of schemes have been presented to enhance the data use to accelerate memory accesses [10], and to enlarge LLC capacities using off-chip memory with representative designs of OSM [1] and Morpheus [11]. Nonetheless, those designs are all based on the precondition that LLC is very scarce, which is mostly controverted by the recent GPU releases. For example, Nvidia aggressively expands L2 [1] cache from no more than 6MB to over 40MB since Ampere. Meanwhile, AMD GPUs have also seen significant advancements in cache architecture to have more levels with much larger capacities. Overall, L2 cache of Nvidia GPUs experiences a rapid growth of 33.4x from Kepler to Hopper, far outperforming the SM count raise of 8.8x. With prolific capacities, LLC is now confronting the challenges to reach high utilization and thus being actively studied for better management [9, 11–13].

Putting together the insufficient L1 and abundant LLC, we are motivated to propose `SMILE` to separate partial LLC as L1D/SMEM extension to host more concurrent threads, i.e., to improve TLP. Overall, `SMILE` preserves a portion of LLC as extended SMEM to allow more cooperative thread arrays (CTAs, a.k.a, thread blocks) to be simultaneously launched onto GPU. To achieve this, `SMILE` first groups SMs into multiple subsets, with each holding a varying number of CTAs to sample the performance changes. For simplification, the additionally launched CTAs are discretely proportional to the native count, and the ratios are generally altering from 10% to 100%. Then `SMILE` enters the sampling phase to collect the number of completed CTAs for each group as performance indication. When

[#]Corresponding author.

[1]Both Nvidia and AMD GPUs name the last level cache as L2. Therefore, we use LLC and L2 interchangeably through the paper, unless otherwise specifically state.

*DAC '24, June 23–27, 2024, San Francisco, CA, USA*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3655906

sampling done, SMILE selects the optimal ratio to partition LLC to permit the desired number of CTAs. For implementation, SMILE involves minor changes on LLC and request addressing, incurring moderate overheads, which are faithfully modeled and evaluated in architectural simulations.

The contributions of this paper are:

- We highlight the observations that GPU TLP can be bounded by the deficient SMEM, and are motivated to divide the increasingly large LLC for SMEM expansion. To the best of our knowledge, SMILE is the first to exploit LLC space to uplift TLP, instead of routinely improving data reuses.
- Through light-weight runtime profiling, SMILE is capable to decide the reasonable partition ratio, and effectively enables extra CTAs to be launched. SMILE can be implemented by slightly augmenting the LLC tags and addressing logics.
- The evaluations on representative applications demonstrate that SMILE remarkably raises the TLP and accelerates the executions, effectively outperforming the state-of-the-arts.

## 2 BACKGROUND AND MOTIVATION

### 2.1 GPU System

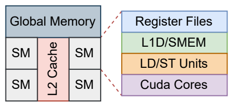


**Figure 1: GPU high-level organization.**

Figure 1 illustrates the high-level architecture of a general-purpose GPU[2], based on recent NVIDIA products. A GPU is composed of multiple compute units, i.e., streaming multiprocessors (SMs). In terms of the memory hierarchy, an SM contains private register files and L1 cache, which is actually a combined L1D and SMEM. Whereas the data cache part is transparently managed by hardware, SMEM is instead software controlled, enabling flexible and dedicated data management to maximize code performance. In practice, the L1D/SMEM split can be dynamically configured to meet varying preferences. The private caches are further backed by a shared L2 cache (or LLC, Last Level Cache) connecting to global memory, which is of HBM or GDDR.

To use GPUs, programmers need to define the parallel portions of applications as kernels, which are then dispatched onto SMs in CTA (i.e., thread block) granularity through device runtime and driver. Before launching a CTA, all the dependent SM hardware resources, including register files, shared memory space, will be scrutinized to ensure that the CTAs can work in parallel. Particularly, threads within a CTA execute in SIMT mode as warps, and they can use SMEM to efficiently communicate and synchronize with each other. The hardware resource utilization is largely determined by the ratio of active threads per SM to the theoretical maximum number of active threads, which is typically 2048 on existing GPUs.

### 2.2 LLC to Boost TLP

Although featuring massive multi-threading, GPUs are frequently showing low TLP, which is typically attributed to shortage of resources like SMEM. To quantify the TLP effects, we run simulations

**Table 1: SM occupancy (*Occu.*) and SMEM usage of applications from [8, 14, 15].**

| App. | Occu.(%) | SMEM(KB) | App. | Occu.(%) | SMEM(KB) |
|---|---|---|---|---|---|
| SPG | 2.08 | 78 | TOP | 6.24 | 32 |
| FG | 2.08 | 69 | LIB | 9.77 | 20 |
| FC | 2.08 | 69 | 2DE | 24.81 | 33.1 |
| GBR | 6.18 | 32 | NQU | 33.30 | 48 |
| CFP | 6.24 | 32 | HBS | 46.31 | 12 |

over a series of benchmarks (details of simulations and applications can be found in Section 4), with results being shown in Table 1. *Occupancy* and *SMEM* are respectively reporting the ratio of concurrently running threads to the theoretical peak and shared memory usage. Clearly, we can see that almost all studied benchmarks are exhibiting low occupancy (2% - 46%), which is inversely proportional to the SMEM metric, agreeing with fact that higher SMEM usage causes less CTAs to be launched and thus lower TLP.

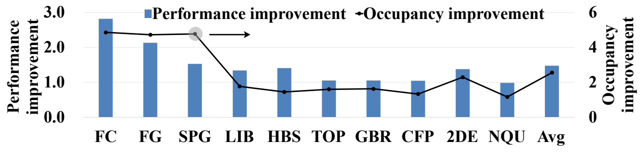


**Figure 2: Performance and occupancy change when doubling SMEM capacity.**

Following the occupancy studies, we next investigate the performance effect with larger SMEM spaces. We reconfigure the simulated GPU with twice SMEM capacity, and collect the IPC statistics to report performance improvement, as being summarized in Figure 2. The figure shows that, most benchmarks benefit from the SMEM doubling, with an average speedup of 1.48× (up to 2.81×). SMEM-hungry ones like FG and FC achieve substantial improvements. Accordingly, SMEM can be very critical, and enlarging SMEM can be promising to improve GPU TLP and performance.

**Table 2: L2 cache usages of applications [8, 14, 15] w.r.t. bandwidth (*bw util.*), percentage of touched lines (*space util.*) and working set.**

| metric | bw util. | space util. | working set(MB) |
|---|---|---|---|
| Average | 8.3% | 71.8% | 4.3 |

Whereas enlarging SMEM is appealing, we are now facing the issue of from where to find space as extra SMEM, especially considering that simply duplicating on-chip L1 can be very expensive and challenging. Fortunately, as aforementioned, recent GPUs now possess huge L2, which is occasionally underutilized. Results in Table 2 list the average statistics about L2 usage, which largely implies the idling of L2. Therefore, it is essential to reduce L2 idling for better data use and performance.

Based on the above observations that SMEM is helpful to solve GPU low occupancy, and L2 on recent GPUs can be underutilized, it is thus natural to integrate both to improve TLP and reduce L2 idling simultaneously. To this end, separating partial L2 becomes a promising solution to expand SMEM.

## 3 DESIGN

In this section, we present SMILE, a TLP improvement technique to enable more concurrently running CTAs by partitioning the shared LLCs as private SMEM extensions.

[2]The paper elaborates using Nvidia GPU architectures and terms, which are generally applicable to other vendors as well.

## 3.1 Overall Architecture

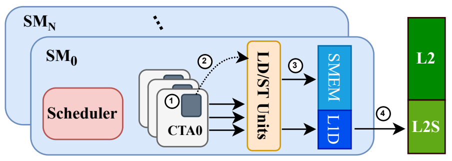


**Figure 3: `SMILE` overall architecture.**

Figure 3 presents the overall architecture of `SMILE`, where the L2 portion separated as extended SMEM is labelled as *L2S*. When being equipped with *L2S*, extra CTAs (eCTAs) could be smoothly launched onto the SMs, given that the SMEM constraint is effectively alleviated. SMEM memory accesses from these eCTAs should be forwarded to *L2S*, physically located in the global L2 cache. To accommodate this, there should be one extra bit for each CTA ① indicating whether this CTA is issued to *L2S* (i.e., eCTA) or not. If the eCTA bit is set, then the LD/ST units will be informed ② to redirect all SMEM accesses from such a CTA to the dedicated portion of *L2S* (datapath ④). Otherwise, the on-chip SMEM accessing datapath ③ is reused for those normal CTAs, just as the default baseline. Note that datapath for normal global memory accesses could also be reused for the eCTA datapath ④, incurring no physical hardware changes.

Since the data logically allocated in SMEM should be available and consistent across the lifetime of a CTA, the data resides on the *L2S* should not be evicted by the cache policy. Therefore, exemption rules are added into the cache policy to avoid cache lines configured as *L2S* being victims when eviction happens. When configuring *L2S* in L2 cache, we also record the physically starting address for each eCTA ①. The starting address together with the SMEM offset (original SMEM access address) are used to translate address for each SMEM access from eCTAs.

## 3.2 Partition Strategy

Although the expanded SMEM capacity allows more CTAs to be issued and executed simultaneously, the pseudo SMEM accesses targeting at L2 cache tend to be slower than the normal SMEM on L1. Thus regarding *L2S* as a free lunch may downgrade performance if the extra issued CTAs are much more than the normal ones. Moreover, employing part of L2 cache as *L2S* inevitably leaves less cache capacity for normal memory accesses, and there will be more off-chip memory accesses produced by extra issued CTAs, burdening the pressure on GPU memory system. In order to reach a balance between the hardware resource utilization and the overall application performance, we propose a runtime profiling-guided (RPG) mechanism to determine the optimal CTA configuration for `SMILE`.

The default SM dispatching policy assigns CTAs to SMs in a round-robin fashion, resulting in a situation where SMs tend to have approximately equal number of active CTAs. The proposed RPG mechanism exploits a non-equal CTA distribution to figure out the optimal runtime configuration. For ease of profiling, we split SMs into multiple groups, and direct scheduler to assign varying number of CTAs to each. Without loss of generality, we basically consider five such groups with different CTA quotas as follow: baseline (no eCTA), $C_1$ (+10% eCTA), $C_2$ (+25% eCTA), $C_3$ (+50% eCTA), $C_4$ (+100% eCTA). During profiling phase, the GPU dispatcher assigns CTAs (including normal CTAs and eCTAs) to SMs in each group with varying limits. For instance, no more CTAs are assigned to SMs in the baseline group since there are inadequate SMEM resources, while 10% more CTAs (eCTAs) could be dispatched to SMs in group $C_1$ whenever other types of on-chip resources are sufficient.

RPG adopts a callback sampling procedure (Algorithm 1) which is called after any CTA is committed to determine the optimal configuration $C_o$ that balances performance and resource utilization. After that, all SMs need to acclimatize themselves to the CTA quota of configuration $C_o$, which we call alignment phase.

**Algorithm 1:** Procedure of RPG profiling phase

```
Input: $N$: throttled number of CTAs during profiling,
       $T[n]$: the number of finished CTAs from each group,
       $S[n]$: the number of rounds each group wins
Output: $C_o$: the optimal runtime configuration
1: if $\sum_{i=0}^{4} T[i] < N$ then
2:     return
3: end if
4: $t \leftarrow \mathrm{argmax}(T[i]), i = 0 \ldots 4$
5: $S[t] \leftarrow S[t] + 1$
6: if $\sum_{i=0}^{4} T[i] \geq 2 * N$ then
7:     Stop profiling
8:     return $C_o$ as the optimal runtime configuration where
       $o = \mathrm{argmax}(S[i]), i = 0 \ldots 4$
9: end if
```

**Profiling phase.** We use two arrays $T[n]$ and $S[n]$ ($n$ denotes the supported count of configurations, and is 5 for the aforementioned CTA quotas), where the former array is to record the number of finished CTAs for SMs from each group since the program begins and the latter one is to record the number of rounds group $i$ is the one with most finished CTAs when Algorithm 1 runs. At the early stage of a GPU program, the number of committed CTAs from each group may change rapidly, so we skip the first few rounds of the algorithm to omit the inaccurate statistics until the total number of committed CTAs from all groups exceeds $N$ (line 1 - 3). In each round, Algorithm 1 will select group $t$ with the maximal overall number of committed CTAs, and increases the corresponding $S[t]$ to mark group $t$ as the champion of current round (line 4 - 5). After another $N$ CTAs are completed (excluding immediate exits at program early stage), Algorithm 1 will return configuration $o$ corresponding to group $o$ that wins the most rounds (line 7 - 9). Once the optimal configuration $o$ is determined, we will never invoke the RPG profiling procedure again (line 7) and next enter the alignment phase.

**Alignment phase.** Once the optimal configuration $C_o$ is determined, all SMs will be ready to take the configuration into effect to have desired number of CTAs to execute. Whereas no extra operations are needed for SMs configured as $C_o$ in the profiling phases, either inflation (increase the CTA quota) or deflation (decrease the CTA quota) should be applied to SMs in other groups. For SMs previously split into group $i (i < o)$, the GPU dispatcher will assign more eCTAs to them and for SMs from group $j (j > o)$, no more eCTAs will be dispatched until the number of active eCTAs drops below the global quota. Normal CTAs will thus be unaffected and dispatched as usual to all SMs. Note that the LLC is split into several slices with fixed size (e.g., 4 MB per slice), allowing RPG to request corresponding proportion of LLC capacity used as *L2S* during the profiling phase based on the number of eCTAs for all groups. The upper limit of *L2S* usage is throttled to 50% of LLC capacity to avoid serious performance downgrade.

## 3.3 Implementation and Usage

**Implementation**: As aforementioned, enabling of extra eCTA launch necessitates hardware enhancements including reserved registers per CTA and modified eviction logic in LLC. `SMILE` adopts two general purpose registers to determine if eCTA or not, and to log the starting address allocated on *L2S* for each eCTA. Note that such registers can be reused from the scalar general purpose registers (sGPRs) already provided in commodity GPUs. As for the LLC space partition and eviction freezing, `SMILE` can simply borrow the existing functionalities on physical GPUs. For instance, Nvidia grants to manage A100 L2 cache in an software controlled fashion [13], and AMD RDNA GPUs permits to adjust cache policy via programming flag bits [16]. The key of `SMILE` falls onto the proposed RPG profiling (Algorithm 1). First, the non-equal dispatch can be realized through setting masks on physical GPUs [17]. And, it is worth noting that the RPG procedure could be invoked asynchronously with the running program by the GPU driver at the host side, causing no performance interference to the primary workload at the device side. While incurring moderate changes and trivial overheads, we aggressively charge one cycle and add 1% extra power as overheads in evaluations.

**Usage**: `SMILE` can be used in a relatively transparent fashion, as the profiling and management are all automatically triggered at hardware level. For backward compatibility, a flag bit can be added to allow manually switch off the `SMILE`.

## 4 EXPERIMENTAL METHODOLOGY

## 4.1 Environment

*4.1.1 Simulation.* We evaluate `SMILE` using Accel-Sim [18] and AccelWattch [19] to model a NVIDIA Ampere-like[3] GPU. Specifically, simulated L2, i.e., LLC, is enhanced to enable space partitions, together with CTA scheduling and request addressing to allow extra CTAs to exploit the enlarged SMEM. Simulation runs are carried out by feeding SASS traces, which are collected beforehand through running the applications on physical GPUs.

**Table 3: GPU configurations in simulator.**

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| Number of SM | 80 | Core clock | 1132MHz |
| Scheduler | LRR | Schedulers / SM | 4 |
| Register File Size / SM | 256KB | L1 Cache / SM | 28KB |
| Shared Memory Cache / SM | 100KB | L2 (or LLC) | 30MB |

*4.1.2 Configuration.* Detailed simulation configurations are as shown in Table 3. For the critical SMEM setting, a maximum 100KB L1 space is reserved as shared memory (i.e., the remaining 28KB as data cache) to facilitate thread concurrency. Such setting aligns with the real GPUs, which allow the unified 128KB L1 to set aside SMEM space ranging from 0-100KB (i.e., 0/8/16/32/64/100KB). For L2, a down-scaled 30MB size is used in simulation to match the maximum of Ampere GPUs.

[3]The modeled architecture is generally based on RTX3070, which is the Ampere architecture officially supported by Accel-Sim. Meanwhile, it can be extremely challenging to simulate extra large workloads to fully cover the massive 120+ SMs and significantly enlarged caches. As such, we choose to adapt the RTX3070 to model a proportionally down-scaled A100, which is roughly 30% less resources than the physical counterparts, on SM, L1 and L2, etc.

**Table 4: Evaluated applications.**

| App | abbr. | App | abbr. |
|---|---|---|---|
| N-queen [14] | NQU | Gemm bias relu [8] | GBR |
| LIBOR [14] | LIB | Conv2dfprop [8] | CFP |
| Hybridsort [14] | HBS | Tensorop [8] | TOP |
| Fused gemms [8] | FG | Sparse gemm [8] | SPG |
| Fused convs [8] | FC | 2Dentropy [15] | 2DE |

*4.1.3 Applications.* To reflect the SMEM-oriented designs, we use multiple workloads involving SMEM usages. As listed in Table 4, the workloads are mainly from two benchmark suites, including Rodinia [14] and CUTLASS [8].

## 4.2 Schemes and Metrics

*4.2.1 Schemes.* To evaluate the effectiveness of our proposed design, we compare `SMILE` against the baseline and prior arts:

— `Baseline`. Default settings on physical GPUs with only partial L1 being used as SMEM and L2 is all for data cache.
— `OSM` [1]. A state-of-the-art approach using off-chip device memory to enlarge the size of shared memory to increase thread-level parallelism. Specifically, the improved version `UCM` is being compared to `SMILE`.
— `SMILE`. Proposed design to separate partial L2 as extended SMEM, and the separated SMEM size is determined by online profiling, as introduced in Section 3.2.
— `Ideal`. L2-based SMEM extension with offline profiling, which serves as the performance upper limit of `SMILE`. For this scheme, optimal L2 partition is exhaustively searched in offline phase, and then gets applied to the application to maximize execution speedup.

*4.2.2 Metrics.* To assess performance, we use the IPC statistics reported by the simulator, and then convert into a normalized speedup to report performance improvement. The associated TLP is measured using the thread occupancy, i.e., ratio of actual thread count to theoretical peak. And, we further employ hit/miss ratio and bandwidth utilization to calibrate the underlying L2 use changes.

## 5 RESULTS AND ANALYSIS

## 5.1 Performance

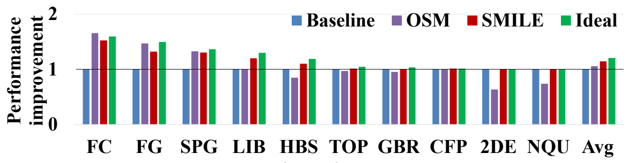


**Figure 4: Comparison of performance improvement.**

We first compare performance improvement of `SMILE` against baseline (no extra SMEM) and `OSM` (with extra SMEM being on off-chip memory). In addition, the `Ideal` with offline profiling is also included to depict the upper bound of `SMILE`. The normalized results are plotted in Figure 4, where workloads are listed in performance descending order from left to right, with *Average* being on the rightmost. From the figure we can see that `SMILE` achieves an average of 14.7% performance improvement across all studied applications, which defeats `OSM` of 5.9%, conveying that SMEM extension on basis of fast on-chip LLC can be much more promising than that of slow off-chip memory. Compared to `Ideal`, our proposed `SMILE` manifests a 5.6% performance gap, which is largely attributed to the contrast decisions between online and offline profiling modes.

On application level, whereas *FC*, *FG* and *SPG* achieve tremendous speedups, others like *2DE* and *NQU* exhibit negligible effects, demonstrating that various workloads are inconsistently behaving on SMEM adjustments, which generally echoes Figure 2 in Section 2.2. Note that for the SMEM-insensitive *2DE* and *NQU*, `SMILE` keeps the baseline configuration by choosing not to extend SMEM, but `OSM` insists on slicing off-chip device memory as SMEM, aggravating memory accesses. Consequently, TLP improvement and performance enhancement are not always identical, for which `SMILE` relies on the proposed profiling to differentiate. Moreover, the workloads *FC*, *FG* and *SPG* benefit more from `OSM`, owing to the fact that `OSM` aggressively combines L1 data cache and shared memory to provide a larger cache for both migrated SMEM and normal global accesses. Despite showing slightly lower speedup on those workloads, `SMILE` circumvents heavy changes on L1 and global memory, and also crucially beats workloads like `OSM` on *LIB* and *HBS*.

## 5.2 Profiling Quality

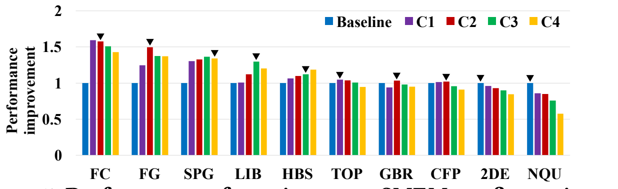


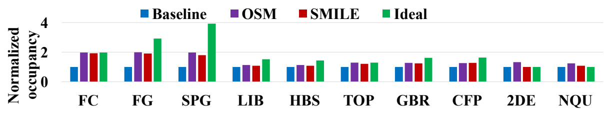
**Figure 5: Performance of varying extra SMEM configurations, i.e., *C1* - *C4*, from which `Ideal` chooses the one resulting in highest speedup. Marked (▼) bars correspond to the eventual decisions made by online profiling in `SMILE`.**

As aforementioned, `SMILE` relies on profiling to decide the proper SMEM extension, which is critical to the overall performance. In order to gauge the profiling quality, we collect the selected SMEM configurations of `SMILE`, and then compare to the ones adopted by `Ideal`. As elaborated in Section 3.2, four extension strategies (*C1* - *C4*) are presented to choose from for `SMILE` and `Ideal`, with the latter exhaustively searching the one resulting in best performance. Figure 5 lists the performance under each strategy in `Ideal`, and also marks (▼) the one employed by `SMILE`.

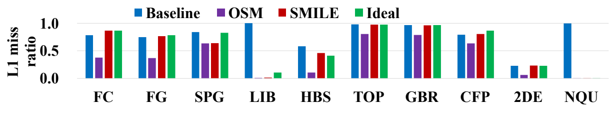
It is clear that workloads, from left to right, are trending to be more conservative on choosing extra CTAs. For instance, *FC*, *FG* and *SPG* select *C2* or *C4*, but *2DE* and *NQU* picks up `Baseline`, i.e., no extra CTAs. Besides, the choices of `SMILE` (marked with ▼) mostly locate on the highest speedup bar, which is just the selection of `Ideal`, confirming the online profiling accuracy. Exceptions occur on *FC*, *SPG* and *HBS*, where the choices of `SMILE` is one grade different from the optimal setting of `Ideal` (i.e., *C1* vs. *C2*, and *C3* vs. *C4*). Such sub-optimal decisions lead to the performance gaps shown in Figure 4.

## 5.3 Occupancy and Cache Changes

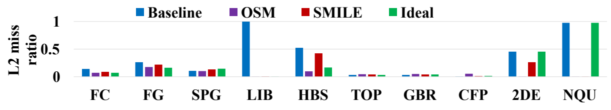
Essentially, `SMILE` trades LLC space for raising thread level parallelism, and is thus undoubtedly impacting the L1 and L2 cache accesses. To have a deeper understanding of the gained performance,



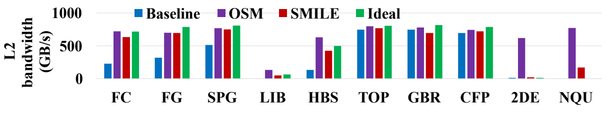
**(a) Occupancy**



**(b) L1 cache miss**



**(c) L2 cache miss**



**(d) L2 cache bandwidth**

**Figure 6: Comparison of occupancy and cache changes.**

we further analyze the associated occupancy, and the underlying cache changes, with results being provided in Figure 6.

Figure 6(a) shows the occupancy (i.e., TLP) of the contending designs. In general, `SMILE` effectively helps increase occupancy over the `Baseline`. Interesting that `SMILE` and `OSM` are mostly comparable in terms of occupancy, while performance apparently differs (see Figure 4). The predominating reason is that `OSM` blindly reserves large off-chip memory portion as the whole SMEM, which actually permits more concurrent threads to run but is at the risk of resource over-subscription. Such side effects can be observed on *2DE* and *NQU*, where `OSM` reports much higher occupancy than `SMILE`, contradicting the speedup shown in Figure 4.

Figure 6(b) - (d) respectively present L1 miss ratio, L2 miss ratio and L2 bandwidth. Be noted that L1 cache, or L1D, here is only the data cache part (i.e., excluding the SMEM), and L2 cache includes both the normal data portion and the separated *L2S* (i.e., the original L2 in GPUs). For `SMILE`, the redirected SMEM accesses onto *L2S* are also filtered through L1D. From Figure 6(b), we see that miss ratios of `SMILE` and `Ideal` are on par with `Baseline`, but generally higher than `OSM`. This happens because `SMILE` and `Ideal` largely keep the default cache organizations, but `OSM` radically expels all SMEM accesses to off-chip memory, and leaves the whole L1 to cache all memory accesses, translating into improved data reuse. Among the workloads, *NQU* and *LIB* have relatively fewer accesses, which can be hidden by the massive shared memory accesses in all SMEM-optimized designs, implying the sharp drops in the figure. Regarding L2 miss ratio in Figure 6(c), `SMILE` and `Ideal` are virtually managing partial L2 as software-control scratchpad, having no data replacements and misses and thus showing lower values. `Ideal` values of *2DE* and *NQU* are identical to the default, as it falls back to `Baseline` on SMEM extension. Bandwidth in Figure 6(d) implies that both `SMILE` and `OSM` are effectively uplifting L2 utilization, indicating the L2's original under-utilization. Specifically, *TOP*, *GBR* and *CFP* retains high utilization in `Baseline`, suggesting the native contention on global accesses instead of SMEM shortage, which coincides with the inefficiency of SMILE. Again, the always high bandwidth of OSM fails to translate into performance improvement, with examples of *2DE* and *NQU*.

### 5.4 Energy

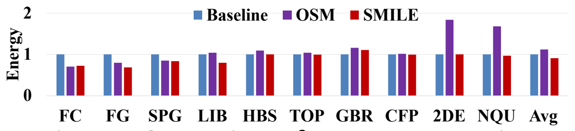


**Figure 7: Comparison of energy consumption.**

To evaluate energy efficiency, we use AccelWattch [19] to collect power statistics and then convert into energy consumption with respect to execution time. The energy results are as reported in Figure 7. From the figure, we can see that SMILE can significantly reduce energy consumption by an average of 9% while OSM increases by 12%. OSM extends SMEM through reserving partial off-chip memory space, which tends to increase data traffic and further the power consumption. Particularly, SMEM-insensitive workloads *2DE* and *NQU* are most energy demanding, which can be attributed to worsen power and lengthened execution time as reported in Figure 4. To the contrary, *FC* and *FG* are favoring SMEM spaces, and thus gain on energy reduction from both OSM and SMILE. Overall, SMILE necessitates less energy on most workloads, conveying the efficiency of LLC-based SMEM extension.

### 5.5 Sensitivity Studies

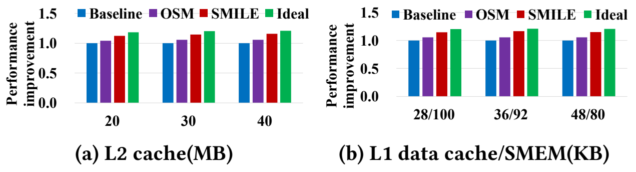


**Figure 8: Performance of varying L1 and L2 configurations.**

Next, we explore the performance sensitivity by varying L2 capacity and L1D/SMEM. Results are separately illustrated in Figure 8(a) and Figure 8(b). As expected, smaller L2 and less allocated SMEM worsen the performance, and slightly narrow down the design space of SMILE. However, SMILE is stably achieving effective performance improvement, revealing that the design is applicable to varying scenarios.

## 6 RELATED WORK

To accelerate GPU executions through better exploiting hardware resources, a set of works have widely explored optimized uses of register files [2, 3], shared memory [5, 20] and interconnects [21, 22], etc. Those works are generally relying on the older GPU generations having very tight LLCs, and are thus orthogonal to our design targeting at GPUs with sufficient cache spaces. Specifically, a myriad of schemes have been presented to better manage on-chip caches, including L1 decoupling [23], LLC extension using core resources [11], LLC partitioning [12] and contention tracking [10]. Particularly, Morpheus [11] argues that LLC is scarce which is based on the previous architecture (5MB L2), and thus strives to work from the opposite direction to extend LLC capacity using GPU core resources. Those approaches are all aiming to improve data reuse, i.e., enhancing data efficiency. Differently, our proposed SMILE is striving to trade LLC space for shared memory, essentially targeting for allowing more threads to accelerate computations.

For boosting thread-level parallelism, CTA scheduling [6, 7] and fine-grained pipelining [4, 8] have been paid much attention. To large degree, those prior arts are complementary to SMILE, which actually permits more CTAs to concurrently run and thus opens up the design space of scheduling and pipelining. As explained in earlier sections, the closest work to SMILE is OSM [1], which enforces all shared memory usages onto off-chip memory and leaves the whole L1 as data cache, necessitating heavier changes than SMILE but may still miss speedup opportunities. To this end, SMILE is expected to be more feasible and applicable to real practices.

## 7 CONCLUSION

This paper proposes SMILE design to separate partial LLC to expand SMEM space. Experimental results demonstrate that SMILE effectively boosts TLP and improves GPU performance.

## ACKNOWLEDGMENT

We thank the anonymous reviewers for their constructive comments and suggestions. This research was supported by the National Key R&D Program of China under Grant NO. 2023YFB3002202, the National Natural Science Foundation of China-#62102465/#62332021.

## REFERENCES

[1] S. Darabi et al., "OSM: Off-chip shared memory for GPUs," TPDS, 2022.
[2] M. Gebhart et al., "Unifying primary cache, scratch, and register file memories in a throughput processor," in MICRO, 2012.
[3] Y. Oh et al., "Linebacker: Preserving victim cache lines in idle register files of GPUs," in ISCA, 2019.
[4] Y.Oh et al., "FineReg: Fine-grained register file management for augmenting gpu throughput," in MICRO, 2018.
[5] P. Sakdhnagool et al., "RegDem: Increasing GPU performance via shared memory register spilling," arXiv, 2019.
[6] D. Tripathy et al., "Paver: Locality graph-based thread block scheduling for gpus," TACO, 2021.
[7] M. Huzaifa et al., "Inter-kernel reuse-aware thread block scheduling," TACO, 2020.
[8] "NVIDIA/cutlass: CUDA templates for linear algebra subroutines." https://github.com/nvidia/cutlass.
[9] S. Zhang et al., "Sac: Sharing-aware caching in multi-chip gpus," in ISCA, 2023.
[10] J. Barrera et al., "Contention tracking in GPU last-level cache," in ICCD, 2022.
[11] S. Darabi et al., "Morpheus: Extending the last level cache capacity in GPU systems using idle GPU core resources," in MICRO, 2022.
[12] X. Zhao et al., "Adaptive memory-side last-level GPU caching," in ISCA, 2019.
[13] Y. Fu et al., "Autoscratch: Ml-optimized cache management for inference-oriented gpus," MLSYS, 2023.
[14] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in IISWC, 2009.
[15] G. Petrov, "2D and 3D image histograms and 2D multistage entropy." https://www.codeproject.com/Articles/15255/2D-and-3D-Image-histograms-and-2D-multistage-entro.
[16] "AMD RDNA2 ISA." https://www.amd.com/system/files/TechDocs/rdna2-shader-instruction-set-architecture.pdf.
[17] M. Chow et al., "Krisp: Enabling kernel-wise right-sizing for spatial partitioned gpu inference servers," in HPCA, 2023.
[18] M. Khairy et al., "Accel-Sim: An extensible simulation framework for validated GPU modeling," in ISCA, 2020.
[19] V. Kandiah et al., "Accelwattch: A power modeling framework for modern gpus," in MICRO, 2021.
[20] S. Pal et al., "Efficient management of scratch-pad memories in deep learning accelerators," in ISPASS, 2021.
[21] A. Li et al., "Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect," TPDS, 2020.
[22] Y. Feng et al., "A scalable methodology for designing efficient interconnection network of chiplets," in HPCA, 2023.
[23] M. Ibrahim et al., "Analyzing and leveraging decoupled L1 caches in GPUs," in HPCA, 2021.