Summary Report

E0-399 Research in Computer Science: Literature Survey

Aman Choudhary (amanc@iisc.ac.in)
Indian Institute of Science, Bangalore, India

1 Research Papers: Summary

1.1 MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability *(2017 ISCA)*

Till recent years, improvement in performance of GPU-based systems was linked to transistor scaling. But, since Moore's law is on decline, cramming more transistors on the die is not feasible. Hence, this paper explores a novel approach to scale GPU performance. Instead of trying to manufacture denser or larger chips, it proposes to integrate smaller chips (called GPU modules or GPMs) at the package-level and expose them as a large logically single-GPU, called *Multi-Chip-Module GPU (MCM-GPU)*.

Since, MCM-GPU is built by integrating compute and memory resources which are distributed over the chip, it leads rise to NUMA behaviour. The paper talks about three methods to tackle this issue: (i) introducing an L1.5 cache to store remote memory access, (ii) scheduling contiguous CTAs to the same GPM in order to exploit inter-CTA data locality and finally, (iii) employing a First-Touch Page Placement strategy. These techniques work together in synergy and help to achieve significant improvement in data locality as well as help in minimizing the inter-GPM bandwidth usage. Overall, MCM-GPU is a major innovation milestone and key to future growth in performance of GPU-systems.

1.2 Combining HW/SW Mechanisms to Improve NUMA Performance of Multi-GPU Systems *(2018 MICRO)*

This paper extends the work carried out in [1]. It cumulatively evaluates the existing state-of-the-art software and hardware mechanisms to improve NUMA performance of multi-GPU systems. Among the software techniques, it talks about mechanisms such as CTA scheduling, Page placement, Page migration and Page replication. It recognises that page migration is not feasible when the pages are shared. On the other hand, page replication bloats the memory footprint of the application, and incurs extra coherence overheads. Of the hardware techniques, the paper evaluates the benefits of caching remote data. Again, caching remote data is not scalable, as the working sets of most GPU applications far exceeds the capacity of the limited cache resource.

Hence, the paper establishes that the existing techniques are not sufficient to counter the effects of NUMA. It then goes on to propose another hardware mechanism, called *CARVE (Caching Remote Data in Video Memory)*.

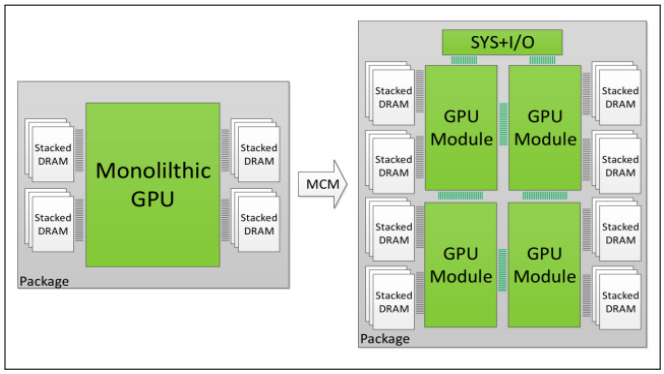

Figure 1: *MCM-GPU: Aggregating GPU modules and DRAM on a single package*

CARVE stores recently accessed remote shared data in a dedicated region of the GPU memory. The data is cached at a finer granularity of 128 B (same as the cache line size) to prevent false sharing which manifests when pages are replicated at page granularity. The paper demonstrates how CARVE boosts the performance of multi-GPU systems and achieves performance that is is within 6% of an ideal NUMA-GPU system (one where all pages are replicated, and coherence overhead is assumed to be zero).

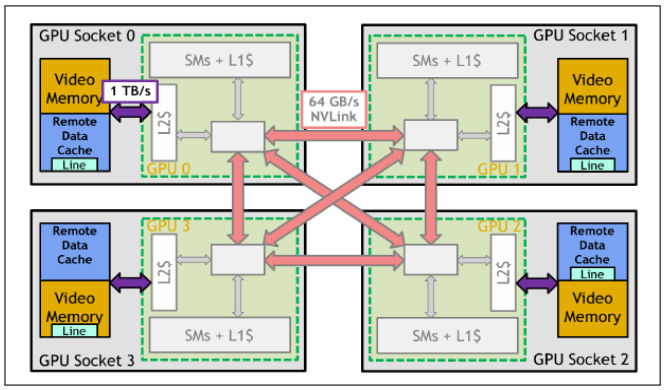

Figure 2: *Architecting a small fraction of GPU memory as a Remote Data Cache (RDC) enables caching large remote data working-sets*

1.3 HMG: Extending Cache Coherence Protocols Across Modern Hierarchical Multi-GPU Systems *(2020 HPCA)*

Till recent years, the GPUs have had a flat single-GPU hierarchy. However, the modern GPU systems are becoming increasingly hierarchical and non-uniform. They consist of a multi-GPU system where each GPU itself, is a collection of smaller GPU modules. Apart from this, earlier GPUs followed the simpler bulk-synchronous programming model

(BSP) which disallowed any data communication among CTAs of active kernels. In contrast, due to an emerging need for less-restrictive data sharing patterns and finer-grained synchronization in GPU applications, recent GPUs have adopted the more relaxed scoped memory consistency models.

As such the existing hardware or software-based cache coherence protocols worked well for the former type of GPU architecture with simpler programming models. However, the later kind of GPUs call for a more relaxed coherence protocol. Specifically the paper proposes a hierarchical hardware coherence protocol that has a two-layered structure, for managing intra-GPU and inter-GPU coherence differently. The proposed protocol equipped with the knowledge of scope semantics, leverages the hierarchical nature of the GPUs, to minimize the inter-GPM coherence traffic, thus improving performance. Overall, it is superior to the existing non-hierarchical alternatives, and achieves performance close to an ideal system with zero coherence overhead.

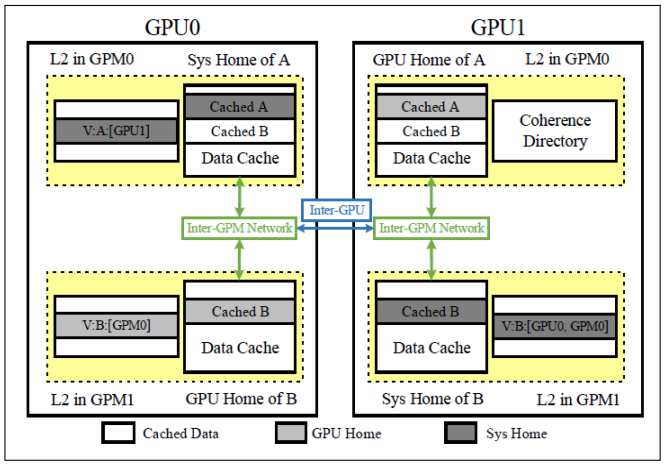

Figure 3: *A hierarchy-aware hardware cache coherence protocol for multi-GPUs*

1.4 Batch-Aware Unified Memory Management in GPUs for Irregular Workloads *(2020 ASPLOS)*

Modern systems employ *unified virtual memory* which relieves the programmer from manual page migration, and lets the application use memory beyond the GPU's capacity. Automatic page migration is supported via *demand paging*, and a crucial part of it is handling page faults. The overhead of handling page faults is very high, due to long latency communication between CPU and GPU over the PCIe bus, and a very expensive *fault handling service routine* performed by the GPU run-time. The goal of the paper is to minimize this cost. As such, the GPU run-time already has a policy in place to handle page faults in large batches to amortize the overhead.

Overall page fault handling time is composed of two components: (i) *fault handling time* and (ii) *page migration time*. The paper proposes two techniques: (i) *Thread over-subscription* and (ii) *Unobtrusive eviction* to minimize these time components, respectively. Thread over-subscription aims at increasing the number of CTAs scheduled to the SMs, by context switching-in inactive ones, once the active CTAs get stalled after incuring faults. As a direct consequence, more faults are generated, resulting in larger batches which further amortizes the fault handling time. The second technique of Unobtrusive eviction aims to remove page eviction off the critical path during page migration. To implement this, it proactively polls the device memory for availability of frames, and makes room by evicting pages, before page migration begins. Hence, migration can always be started instantly without delays. In summary, the proposed optimizations provide an average speedup of 2x over the state-of-the-art mechanisms.

1.5 The Locality Descriptor: A Holistic Cross-Layer Abstraction to Express Data Locality in GPUs *(2018 ISCA)*

A major factor that dictates GPU performance is how efficiently, the memory subsystem is utilized. The key to judiciously use the memory is to improve the data locality, that is make data easily available to the compute. The paper talks about two kinds of locality: (i) *Resuse-based locality* which is reuse of data within the application in the cache hierarchy, and (ii) *NUMA locality* which is placement of data close to the computation in NUMA systems. The problem is that although, the programmer has at his disposal, the means to explicitly express parallelism that exists in the application, no such mechanism exists to convey the data locality.

This lack of expressiveness prevents the programmer to write code that can get the best performance out of the memory hierarchy as software does not have easy access to hardware optimization techniques. On the other hand, hardware-only architectural techniques are often sub-optimal as they miss key higher-level program semantics that are essential to effectively leverage data locality. Hence, the paper proposes the *Locality Descriptor* to bridge this gap. It is a cross-layer abstraction to explicitly express and exploit data locality in GPUs. In this technique, the programmer can annotate the data structures used in the application and describe the pattern of how the data is being accessed by the CTAs. Overall, the paper demonstrates that using Locality Descriptor, significantly improves performance and is capable of exploiting both types of data locality efficiently.

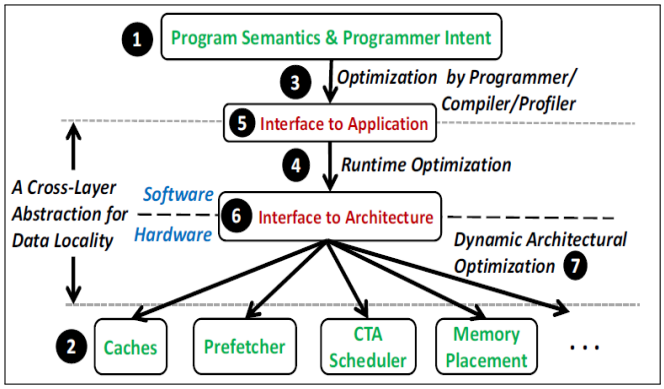

Figure 4: *An overview of the Locality Descriptor abstraction*

2 A holistic view of the papers

Historically, the improvement in GPU performance has been tightly coupled with transistor scaling. However, the recent trends suggest that Moore's law is on the decline. We cannot make denser or larger chips without compromising the yield and cost of chips. Hence, the research community has recognised the need to look beyond single-GPU systems. As a result, the focus has shifted to multi-GPU architectures. The latest trend goes one step further and introduces another layer of hierarchy in the multi-GPU system. Now, each GPU in the multi-GPU system is itself a MCM-GPU, that is a collection of GPU modules integrated at the package level using high speed interconnect technology.

However, a hierarchical multi-GPU design comes with its own set of problems. Now, the programmer needs to be aware of the multi-chip architecture, and tailor the application accordingly to make the best use of GPU, thus, increasing the programmer effort. However, this is not the most pressing concern. The major challenge with the multi-GPU architecture is the manifestation of NUMA behaviour, which is a significant performance bottleneck. Paper [1] and [2] are primarily focused to mitigate this issue. Paper [1] talks about caching remote data in L1.5 cache, scheduling consecutive CTAs to the same SM, and employing a first-touch page placement policy. Paper [2] which was released just a year later, by the same group of researchers, goes on to declare the above set of optimizations along with other techniques such as page migration, and page replication, as sub-optimal. They then propose to extend the idea of caching remote data at a larger magnitude then before, by carving out a small chunk (roughly 3%) of the GPU memory per chip, to serve as a Remote Data Cache (RDC). This significantly reduces the inter-GPU traffic and acieves an impressive performance.

Paper [2] also sheds light about the coherence mechanism that would be required to maintain consistency among the data cached in multiple GPUs. They try to extend the existing software coherence protocol, but due to its poor performance, move on to an improvised version of the existing hardware cache coherence protocol. Paper [3] which was also the work of the same group as [1] [2] addresses the fact that, owing to the increasing hierarchy in GPU systems which follow the more relaxed scoped memory consistency models, there is a need to review the existing software and hardware coherence mechanisms. It proposes a hierarchical coherence protocol tailored for a hierarchical multi-GPU system. The outlined solution significantly reduces coherence traffic by maintaining consistency within the scope provided by the programmer, and relaying coherence messages in an hierarchical fashion. In summary Paper [1], [2] and [3] talk about scaling GPU performance by following a hierarchical multi-GPU approach. They also address the different problems that are associated with such systems.

Paper [4] talks about an orthogonal issue i.e. of page fault handling in a unified virtual memory setting, and suggests methods to minimize the associated overhead, using techniques like Thread Over-subscription and Unobtrusive eviction. Finally Paper [5] talks about improving data locality, by equipping the programmer with appropriate abstractions to explicitly express data use patterns in the application. It details how with the available program semantics, the hardware can make more informed decisions regarding the choice of optimizations. The proposed solution can also work in synergy with the above set of optimizations to mitigate NUMA on multi-GPU systems by localizing compute and data.

3 Tentative Action Plan

After reading the above set of papers, one direction of work that looks plausible to me, is to look further on how one could optimize for the problems that come with multi-GPU systems. To be specific, I could aim to look for novel techniques or optimize existing ones, with the goal of improving data locality and further minimizing inter-GPU communication, and suppress the effects NUMA behaviour.

4 References

[1] *MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability (2017 ISCA)*

[2] *Combining HW/SW Mechanisms to Improve NUMA Performance of Multi-GPU Systems (2018 MICRO)*

[3] *HMG: Extending Cache Coherence Protocols Across Modern Hierarchical Multi-GPU Systems (2020 HPCA)*

[4] *Batch-Aware Unified Memory Management in GPUs for Irregular Workloads (2020 ASPLOS)*

[5] *The Locality Descriptor: A Holistic Cross-Layer Abstraction to Express Data Locality in GPUs (2018 ISCA)*