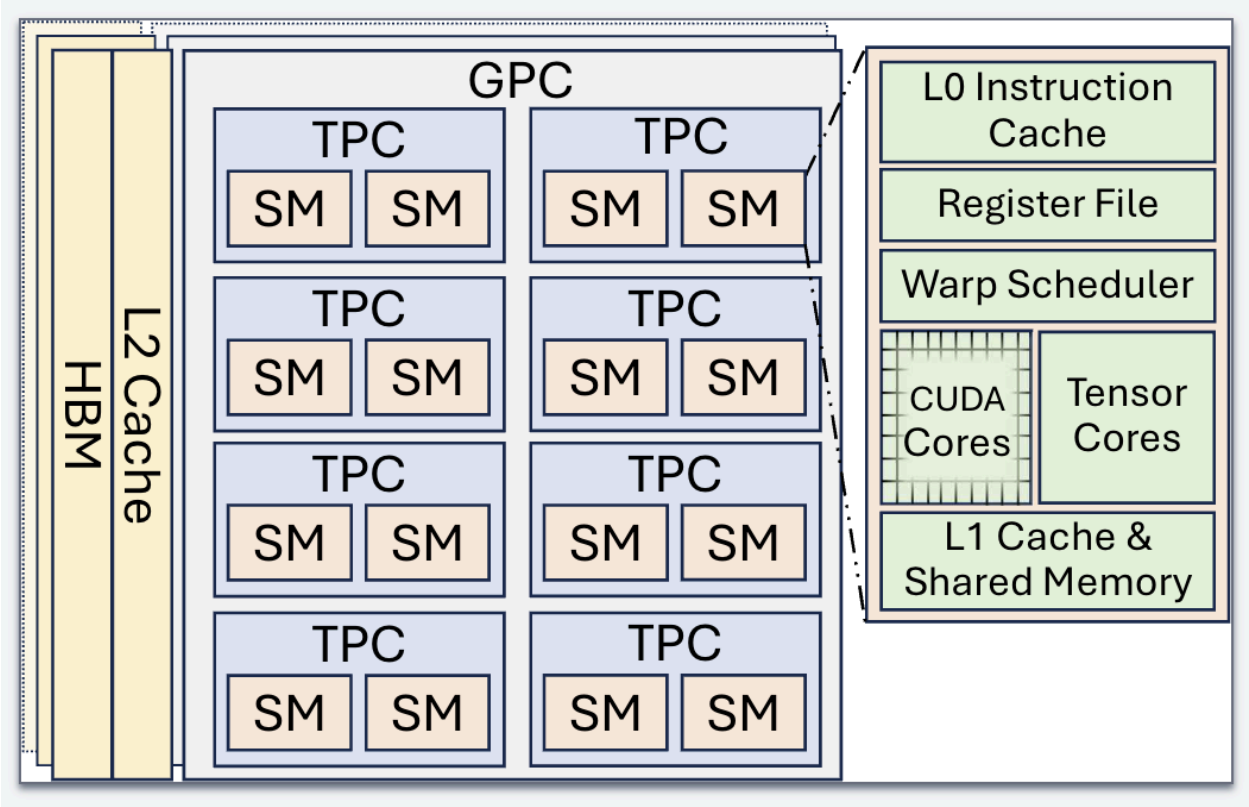
NYSRG: GPU Sharing, Nov 10 2024

GPU Architecture and Programming Model:



Higher Level Overview of GP GPU (General Purpose Graphical Processing Unit) Architecture

Readings:

- Task Parallelism vs Data Parallelism
- Understanding GPU Architecture
- Understanding GPU Programming Model
- (Optional) Comparing CPUs vs GPUs for HPC at a (very) high level

Key Takeaways:

- GPU's focus on SIMD (i.e. Data Parallelism)
- Their Programming model revolves on around having several hundreds of cores with thousands of threads that can run simple arithmetic operations as compared to a CPU that has a much fewer but larger Cores that are more "intelligent"
- Memory hierarchy: An SM (and its CUDA cores within) have a fast L1 cache/shared memory, but inter SM communication is possible via GMEM or L2 cache but this much slower.

- Today GPU's (at least with NVIDIA) have branched into GP GPU's for more AI/crypto mining driven workloads (like in the figure above, with Tensor Cores) and into Gaming GPUs (with RT Cores) built for image processing/rendering.
- GPU Kernels are functions executed on a GPU **(have nothing to do with OS kernels lol)**

Executing Multiple Applications on a GPU:

Background/Context:
GPGPUs in production are often under-utilised with companies like Microsoft reporting a median util of ~50%. With the cost of GPU's and their expensive energy consumption, this is not ideal. Hence, there is a demand for research to improve this.

NVIDIA's Built-in GPU Sharing Methods:

- Time slicing: Timeslicing is enabled by default on NVIDIA GPUs and tasks share the entire GPU in a round-robin fashion. In each given time quanta which lasts several ms, a task has exclusive access to the entire device. In contrast to CPU context switching, GPU context switching is order of magnitude slower due to the large number of thread contexts and GPU memory that needs to be saved.
- Multi Process Service (MPS) and CUDA Streams (Focus mainly on CUDA Streams and MPS, can go through Hyper Q if we have time)
    - NVIDIA Docs MPS
    - NVIDIA Docs CUDA Streams (Note NVIDIA now also offers priority streams)
- (Optional) In Depth Explanation of GPU Streams
- Multi Instance GPU (MIG)

Key Takeaways:

- Time slicing shares the GPU in a round-robin fashion, giving each task exclusive access for several milliseconds. A major limitation of these methods is that they execute only one job at a time, leading to low utilization.
- MPS allows for the concurrent usage of the GPU by multiple tasks at once by merging multiple GPU contexts into one shared context by taking advantage of Hyper-Q and CUDA Streams. This can yield greater throughput but often leads to unpredictable performance interference between applications.
- MIG partitions the GPU's compute and memory resources along GPC boundaries, providing strong hardware isolation. However hardware resources (compute and memory) cannot be dynamically tuned.

Current State of the Art for GPU Sharing

- Transparent GPU Sharing (TGS)
    - Transparently interposes libcuda to allow multiple applications to temporally share a GPU by differentiating them by high and low priority

    - Based on the rate at which kernels are high priority (production) launched, the launch rate of the low priority (opportunistic) jobs is then decided to make best possible use of the device.
    - To handle memory oversubscription, TGS takes advantage of NVIDIA's UVM model and cudaMemAdvise to specify the behavior of high and low priority applications' allocated memory to ensure isolation.
- REEF
    - Also like TGS, it is transparent and differentiates between high and low priority applications but differs from TGS by improving resource util via spatial partitioning.
    - Introduces Dynamic Kernel Padding to pad high priority jobs with lower priority kernels to improve utilization. They also introduce a preemption module to interrupt any low priority tasks running on a GPU to avoid contention with a high priority task.
- Discovering how to give Users/Programmers control over spatial partitioning
    - This group discovered a "mysterious field" being added to kernels when launched on a GPU and found out that was what determined which SMs each kernel was mapped to.
    - They produced libsmctrl which allows you to set masks that specify which TPC you want a kernel to run on.

More Interesting and Related Works

- Orion: This was very promising but is not transparent (Required modifications to user applications)
- Antman: Similar to the above ^
- Tally: This is very similar to the work my group submitted to ASPLOS'25 (scarily so) but very cool.
- Clockwork

Glossary

- SM (Streaming MultiProcessor). Used interchangeably with CU (Compute Units), the equivalent on an AMD GPU
- GPU Kernel: Function launched on a GPU that contains user specified params such as number of threadblocks, threads etc.
- SIMD: Single Instruction Multiple Data
- High Priority Applications: Used interchangeably with Production, latency critical (LC) and real time (RT) jobs
- Low Priority applications: Used interchangeably with Opportuinistic and best effort (BE) jobs.

Additional Info

NVIDIA GPU Memory Model:

- Unified Virtual Memory Model
    - Understanding the effects of memory oversubscription and page faults
    - An attempt to improve resource util during a page fault