Resources partitioning and latency hiding Part 1:

We will now look at the resource limitations of the **SM** and how exactly this scheduling works. The **local execution context** of a **warp** mainly consists of the following resources:

* program counters
* registers
* shared memory

The **execution context (instructions executed)** of each **warp** processed by a **SM** is **maintained on-chip during the entire lifetime** of a **warp**. Therefore, **switching from one execution context to another has no-cost**. This is expected since GPUs are designed to execute thousands of **threads** and if the **context switching took a long time, the overall execution will be delayed**.

However, this **zero-cost context switching is achieved** by **making GPU threads more lite weight and limiting its capabilities**. Out of above mention resources, the **number of registers** and **shared memory** can be **directly controlled by the programmers**.

Each **SM** has a set of **32-bit registers** **stored in a register file** **that are partitioned among threads** and a fixed amount of **shared memory** that is partitioned amount **thread blocks**. The number of **thread blocks** **warps** that **can simultaneously reside** on a **SM** **for a given kernel depend on:** **Number of registers** and **amount of shared memory** available on the **SM** and required by the kernel.

EX:

Diagram

Description automatically generatedAs shown here, if each **thread** for a given **kernel** **consumes more registers,** **fewer warps can be inside of a** **SM**. If **you can reduce the number of registers** a **kernel** **consumes, more warps will be processed simultaneously**.

Diagram

Description automatically generatedLikewise, if the **thread block** **consumes more shared memory** **fewer** **thread blocks** **are processed simultaneously by a** **SM**. If you can **reduce the amount of shared memory used by each thread block,** then **more thread blocks can be processed simultaneously**.

What this means is that **resource availability** **generally limits the number of resident thread blocks per SM**. The **maximum** **number of registers** and **amount of shared memory** per **SM** vary with depending of a device compute capabilities. If there are **insufficient** **registers** or **shared memory** on each **SM** to **process at least one** **thread block**, then the **kernel launch will fail**.

The table lists **some of the technical limitations** like:

* Table

  Description automatically generatedMaximum number of concurrent blocks can reside in a **SM**
* Maximum concurrent **warps** per **SM**
* Number of 32 bit registers per **SM**
* Maximum number of 32 registers that can be used by single **thread**
* Shared memory for **SM**

Some of these **values** **can be retrieve** with **CUDA** **get device properties function** and others **are listed in particular microarchitectures documentation**.

Table

Description automatically generatedEx:

In a device with **compute capability 3 or higher** we **have** **64000 32-bit registers** per **SM**. These **registers** should be **shared across** all the **thread blocks** **running** in that particular **SM**.

Table

Description automatically generated

If you look further, you can see that one **thread** can **only use** **255 registers only**. If we use more than that in our program, **those excessive registers are going to spill over to memory** called **local memory** and that **will reduce the performance of the program greatly**. We can **optimize the performance of a kernel** **depending on the device that is going to execute on**.

Diagram

Description automatically generatedLet's now look at categorization of **blocks** and **warps**. A **thread block** is called an **active** **block** when **compute resources** such as **registers** and **shared memory** **have been allocated to it.** **Warps** in such an **active** **block** is called **active** **warps**. **Active** **warps** can be further classified in to following three types:

* **Selected** **warps** = An **active** **warp** that is actively executing
* **Stalled** **warps** = An **active** **warp** no ready for execution
* **Eligible** **warp** = An **active** **warp** ready for execution but currently waiting for execution

The **warp** scheduler on a **SM** **selects** **active** **warps** **on each execution cycle and dispatch them to *execution unit***. A **warp** that is actively executing is called **selected** **warp**. If an **active** **warp** is **ready for execution but currently waiting to execute it is** an **eligible** **warp**. If a **warp** is **not ready for execution it is** a **stalled** **warp**.

A **warp** is ELIGIBLE FOR EXECUTION IF BOTH OF THE FOLLOWING CONDITION IS MET:

* 32 **CUDA** **cores** should be available for the execution
* All arguments to the current instructions should be ready

The **number** of **active** **warps** at any time **from launch to completion on microarchitectures** **with compute capability 3 or above** **must be less than or equal to the architectural limit of 64 concurrent warps**. If **warp** **stalls**(stops), the **warp** **scheduler picks up an eligible** **warp** to **execute in its place**. Because **compute resource is partition among** **warps** and **kept on chip during entire lifetime** of a **warp**, **SWITCHING WARP CONTEXT WILL NOT COST ANY EXECUTION CYCLE.**