Spring 2019 CS695

Software-based multiplexing of GPU

Neeraj Kerkar (18305R002)
Shah Rukh Hussian (18305R009)

Project description

Context:

- Kernel: routing compiled for GPU
- Once kernel is executing on GPU pre-emption is not possible
- GPU resources can't be time shared
  when executing other kernel.

Goal:

- Optimal scheduling algorithm and number of slices of kernel to maximize the total throughput, avg turnaround times of the kernels

Checkpoints of Progress:

- Performing Kernel slicing and schedule these slices using round robin.
- NVIDIA Visual Profiler to compare performance of slice and non slice kernels.
- Explore different scheduling algorithm.
- Estimate number of slices in each kernels.

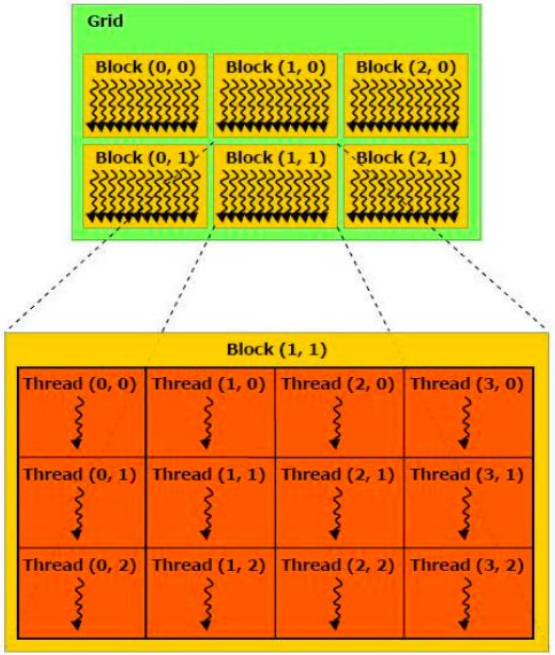


Figure 1. CUDA thread hierarchy. Image Source- [http://www.nvidia.com/object/cuda home new.html]

Project components

We will provide a software framework to be used by developer.

- **Kernel Slicing**:The framework splits kernel call into many kernel calls of small size.It maintains state like #blocks remaining for each of the kernel.
- **Scheduling**: Implement different scheduling algorithms such as Round Robin and Shortest Job First.
- **Performance Testing**: Performance study based on kernel throughput in slice and non slice kernel.

Approach details

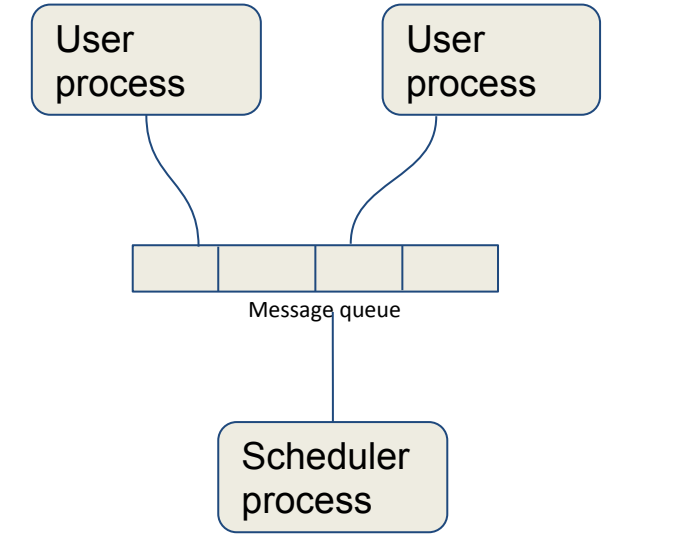
We implemented a macro for kernel slicing:

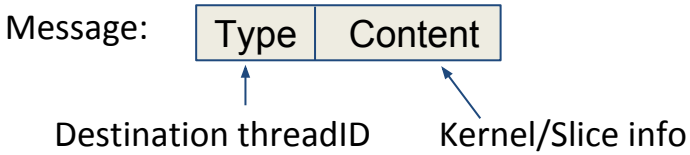
Instead of
kernel<#blocks>(args)

Use macro
SLICER(kernel, #blocks, args)

```
SLICER(kernel, #blocks, args):
        register_kernel_with_scheduler(#blocks)
        while(kernel not done):
                #sblocks = get_slice_info_from_scheduler()
                kernel<#sblocks>(args, blockOffset)
                tell_scheduler_slice_done()
```

We used linux message queue IPC to talk with Scheduler





More approach details

Scheduler process : central entity to coordinate slicing

Scheduler loop:

```
While (msg = get_msg_from_queue()):
        if(msg is REGISTERATION):
                add new kernel_struct
        if(msg is SLICE_FINISHED):
                Choose next slice from some kernel in list
                Tell user process to run slice
```

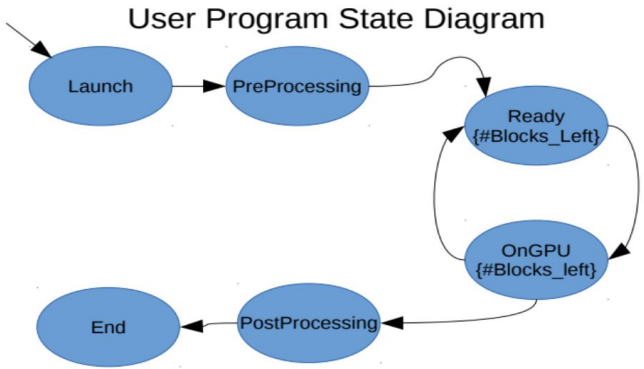


Figure 3.User Program State Diagram [GPUScheduler : User Level Preemptive Scheduling for NVIDIA GPUs]

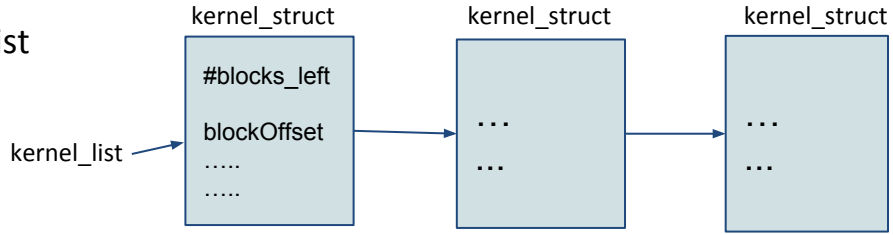


Figure: Scheduler maintains kernel state in kernel_struct

Evaluation plan

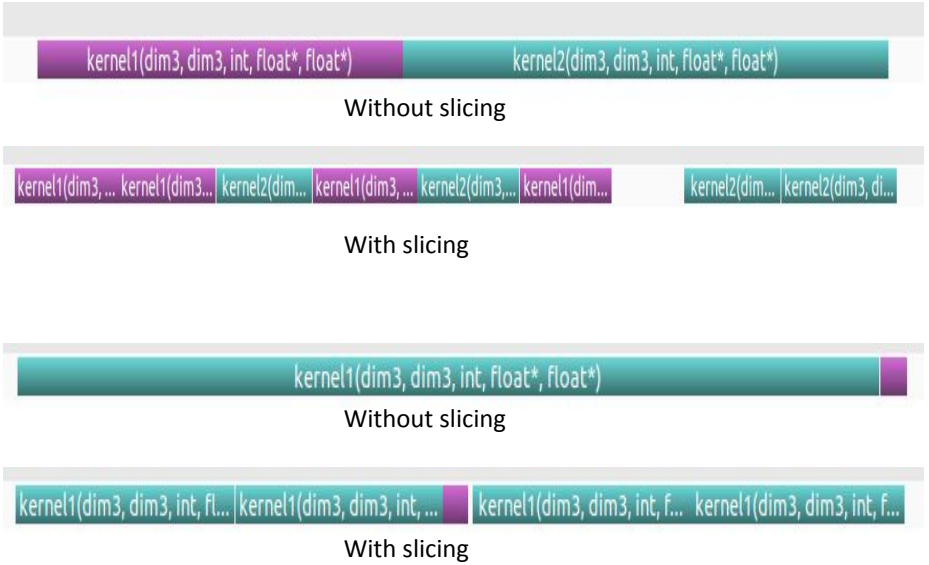
Correctness: output of matrix/vector add kernels were same with and without slicing

Performance:

- Evaluation of slicing overhead by varying slice size of kernel and find appropriate number of slices
- Comparison between different scheduling algorithms.
- Performance study on kernel throughput in slice and non sliced scenarios

Results so far: (Using NVIDIA Visual Profiler)

- Profiled code with/without round robin policy
- Kernel1, kernel2 launched simultaneously by different threads



Timeline for what remains

- Performance evaluation on various scheduling algorithms.
- Slicing kernel in appropriate number of slices.
- Performance testing on sliced and non sliced kernel.