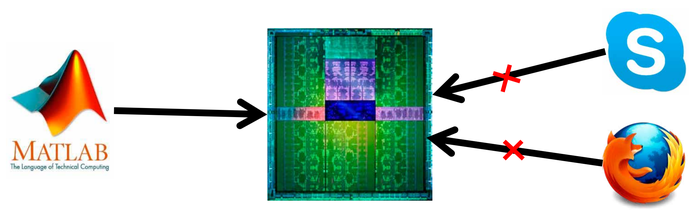
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Hardware Support for GPU Multiprogramming

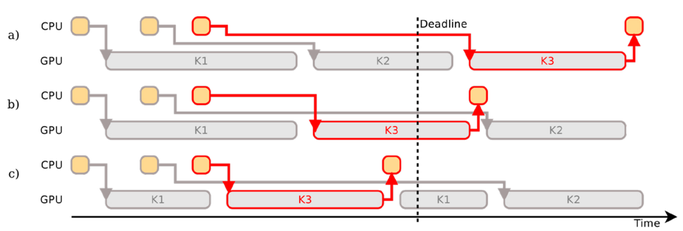
Ivan Tanasić[12], Isaac Gelado[3], Javier Cabezas[12], Alex Ramirez[12], Nacho Navarro[12], Mateo Valero[12]

Barcelona Supercomputing Center[1] Universitat Politecnica de Catalunya[2] NVIDIA Research[3]

- The number of GPU accelerated applications is growing – look around you!
- Some of these are HPC applications (isolated execution)
- Others are interactive user applications, running on mobiles, PC or the cloud (multi-programmed environments)
- GPUs (currently) lack support for multiprogramming – in the traditional (general purpose operating system) sense
- As sharing increases, user experience will decrease ☹



- The biggest obstacle for multiprogramming GPUs are
  -Non-preemptive kernel execution (improves throughput)
  -Non-preemptive data transfers
- We simulate a Kepler – like GPU, extended to support two preemption mechanisms (*draining* and *context switch*) and two scheduling policies:
  -*Preemptive Priority Queues* (time multiplexing)
  -*Dynamic Spatial Sharing* (space sharing)
- Simulation results show big improvements in responsiveness and fairness, at the expense of average throughput degradation (up to 50% with 8 processes)



Benefits of preemptive scheduling on latency sensitive kernels

- Completion time of the high priority kernel (K3) , when co-scheduled with low priority kernels (K1 and K2) , with FCFS scheduler (a), non-preemptive priority scheduler (b), and preemptive priority scheduler (c).

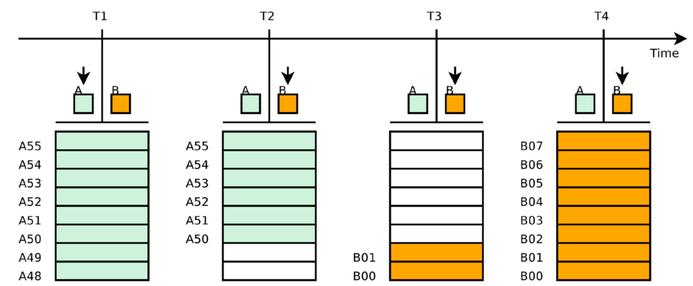


Illustration of the *draining mechanism*

- Preempts the execution of a kernel (illustrated on one SM) by letting all the active (already running) thread blocks to finish their execution, without issuing new work to the SM.
- It is a low overhead scheme with intuitive implementation on modern GPUs, but it's performance depends on the granularity of the thread blocks.

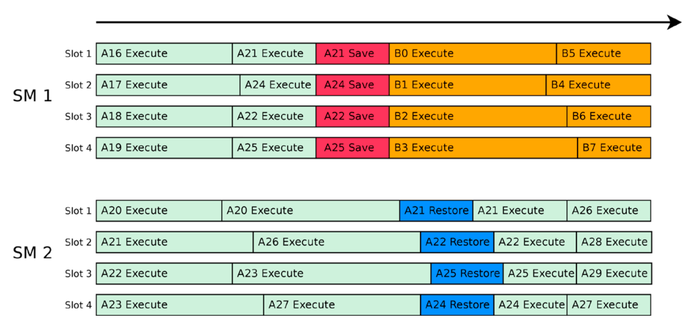
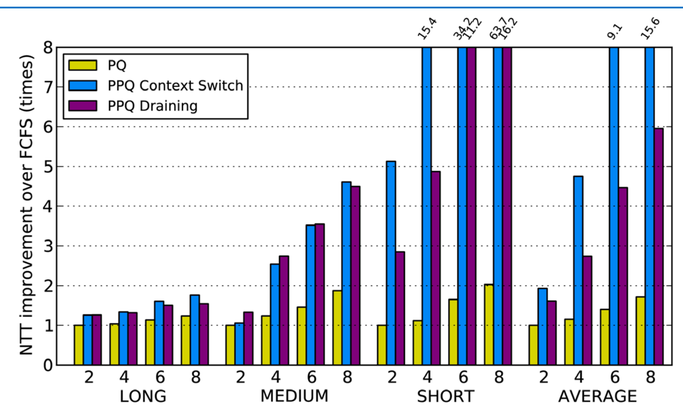


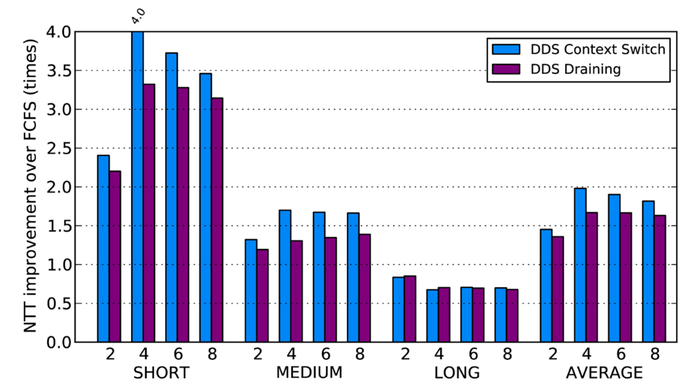
Illustration of the *context switch* mechanism

- Preempts the execution of a kernel (illustrated on one SM) by stopping the execution of thread blocks and saving the execution context to the off-chip memory (restored on reissue).
- Requires additional hardware support and incurs overhead off saving and restoring the context , but it's performance does not depend on the execution time of thread blocks.



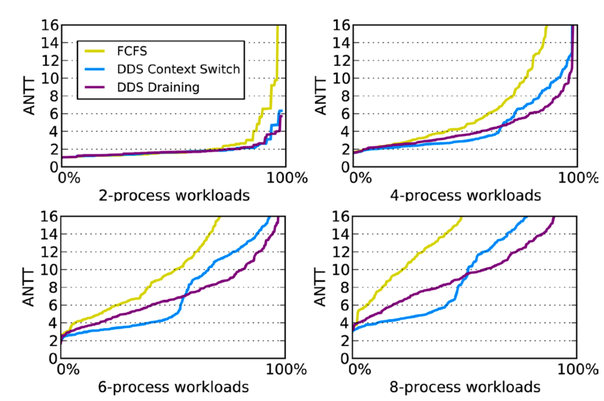
Normalized Turnaround Time improvement with *Preemptive Priority Queues* (PPQ) scheduler

- Improves the NTT over *First Come First Served* (FCFS) and non-preemptive PQ.
- Benefits increase with the number of processes (2 to 8) competing for the GPU .
- Benchmarks are grouped by their execution time.



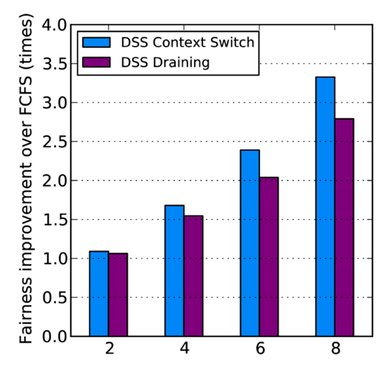
Normalized Turnaround Time improvement with *Dynamic Spatial Sharing* (DSS) scheduler

- Improves the NTT over FCFS by allowing kernels from multiple programs to execute on different sets of SMs, preventing the starvation of SHORT benchmarks.



Average Normalized Turnaround Time of workloads with *Dynamic Spatial Sharing* (DSS) scheduler

- ANTT shows that performance of two preemption mechanisms depends on the workload.



Fairness Improvement with *Dynamic Spatial Sharing*

- Obtained through equal resource allocation

Conclusions

- Necessary to sustain the growing ecosystem of GPU accelerated applications
- Improves some metrics (responsiveness and fairness) at the expense of others (throughput)
- Where is the right tradeoff?
- How to minimize drawbacks?