UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Deadline-based Scheduling for GPU with Preemption Support

N. Capodieci, R. Cavicchioli, M. Bertogna, A. Paramakuru.

University of Modena and Reggio Emilia

NVIDIA Corp.

Outline

- Motivation
- Preemption on GPUs
- NVIDIA Baseline interleaved scheduler
- Our Prototype: EDF+CBS on GPU
- API extension
- Experiments and Results

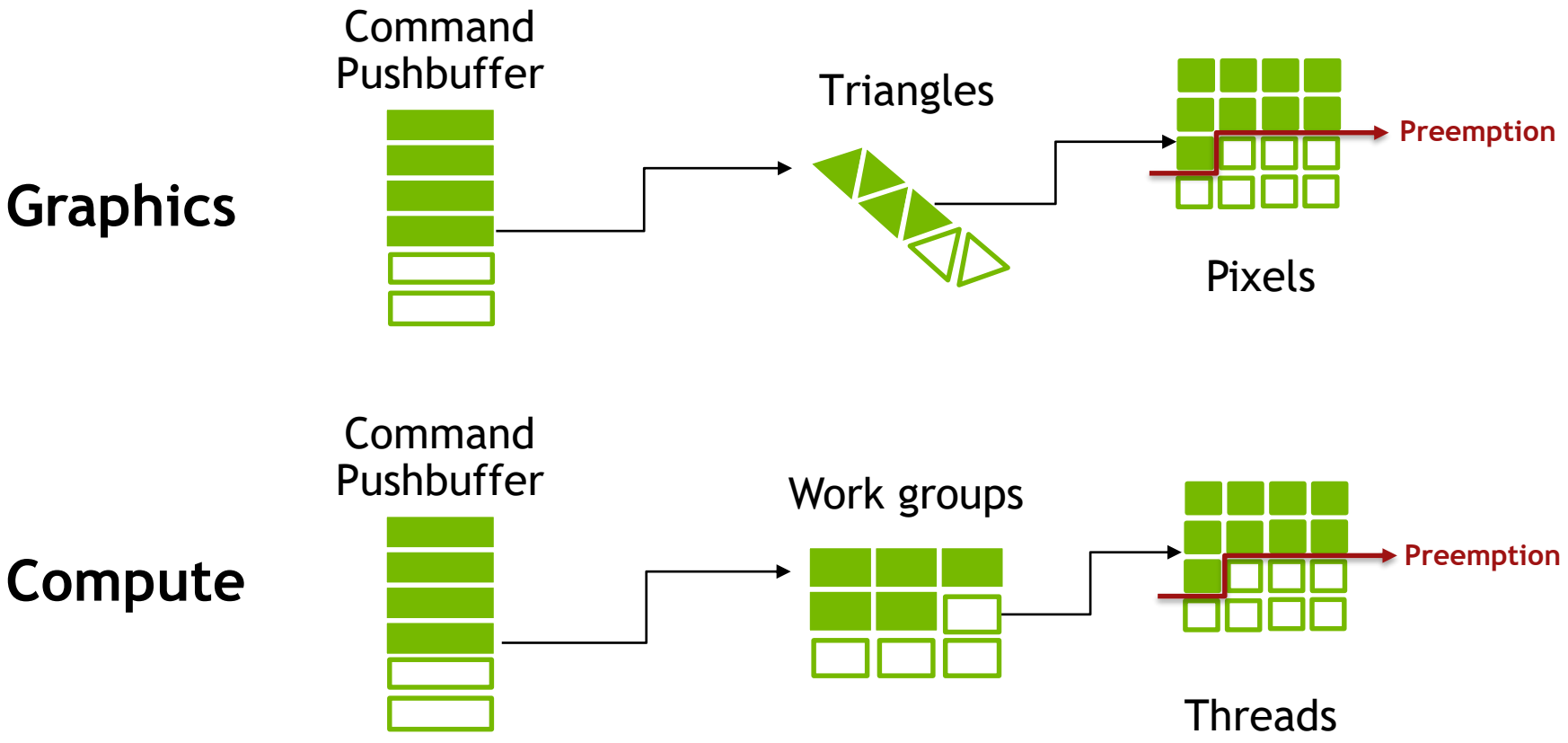
Motivation

- GPUs are heavily used in HPC applications
- GPUs' massive parallel architectures can be used for ADAS within embedded (low power) systems
- New challenges:
  - Safety
  - **Real Time constraints**
  - **Mixed Criticality**

Challenges for GPU and Real Time requirements

- Limited Preemption functionalities on GPUs:
  - Strong limits to applicable scheduling mechanisms
- Preemption granularity too coarse, or even absent.
- A GPU is a composition of independent engines:
  - **Commands** and **Data** are constantly streamed CPU->GPU
  - **Common APIs** (CUDA, OpenGL, OpenCL ...) are not designed to support RT scheduling
  - **Very recent research area…**

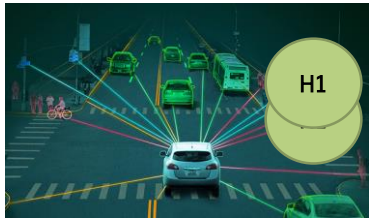
Preemption on GPUs (From Pascal Arch.)

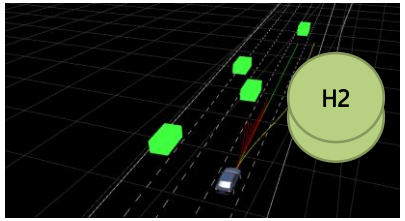


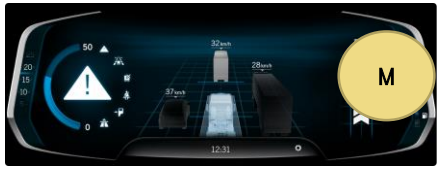
NVIDIA Interleaved scheduler

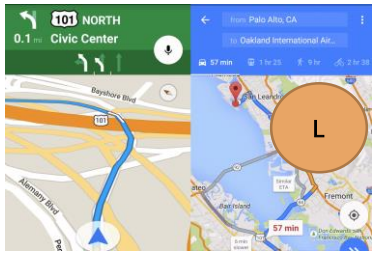
- Each application maps to one or more *channels*
- GPU Host Scheduler snoops on a Runlist of channels
- Each channel is characterized by a *timeslice* and a *priority / interleaving level*
- Runlist is built from this information

NVIDIA Interleaved scheduler

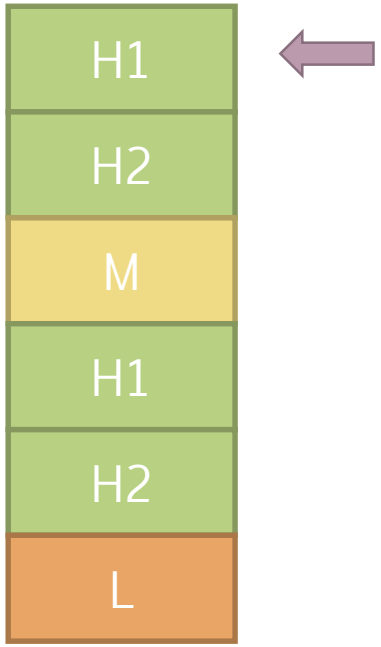


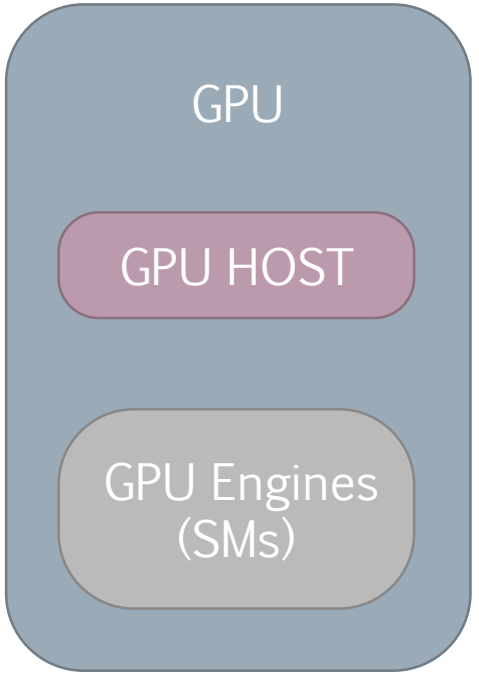






RUNLIST

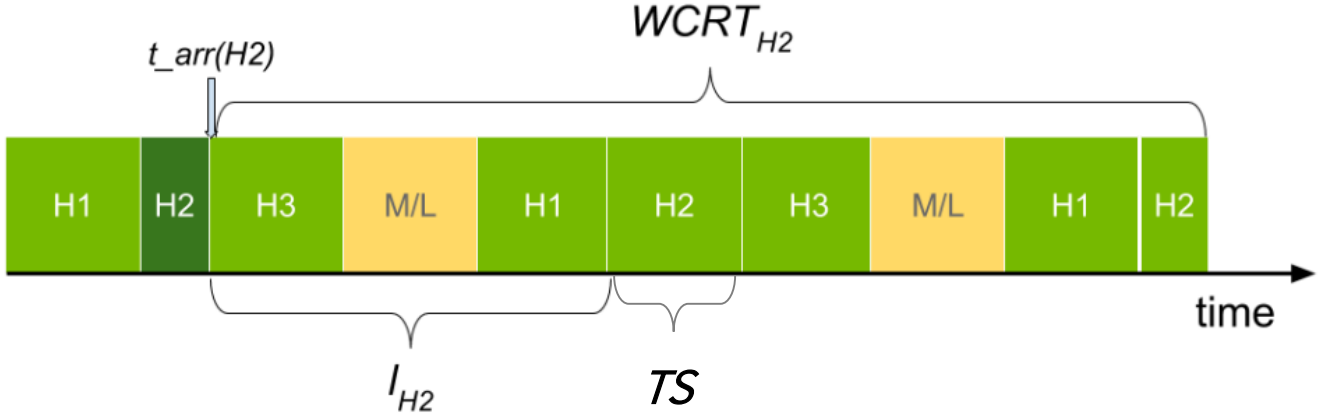




Schedulability analysis of NVIDIA interleaved

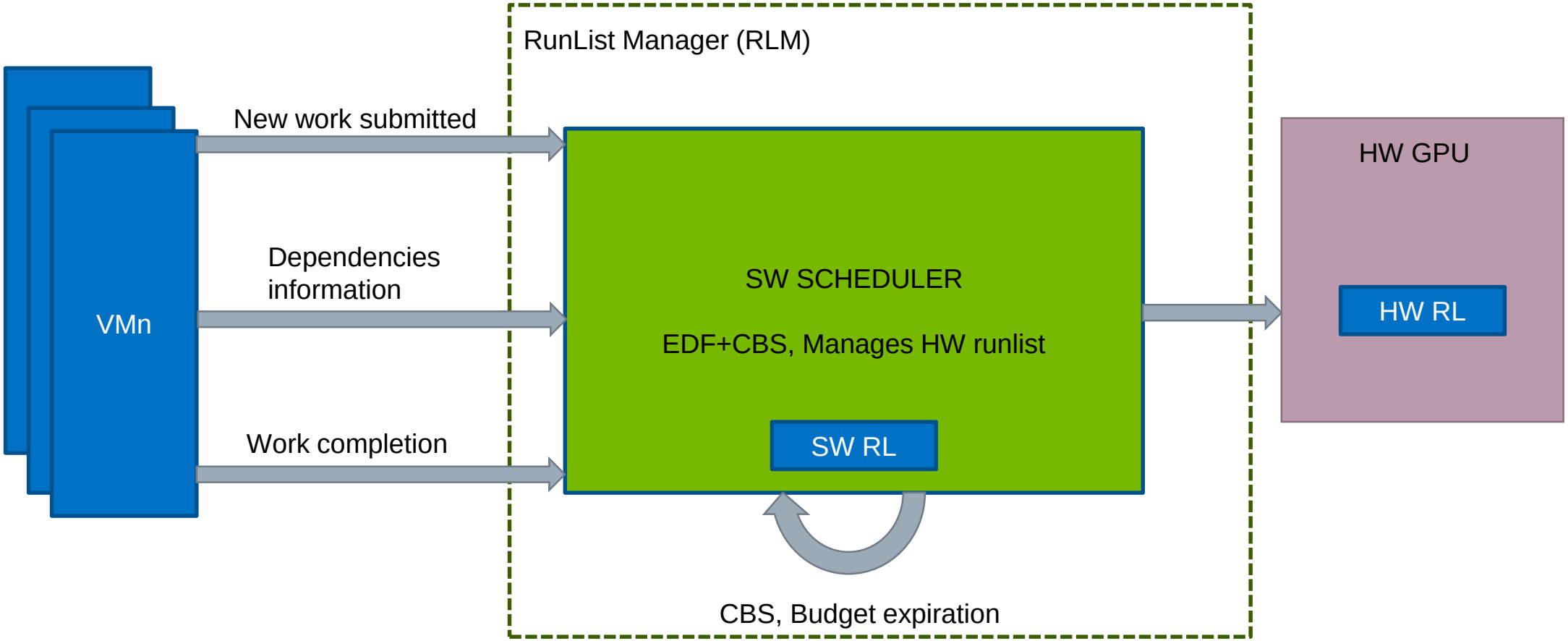
$$l_i = \sum_{\substack{j=1 \\ j \neq i}}^{NR} \overline{TS_j} + TS_{M/L}$$

$$R_i \leq \left\lceil \frac{C_i}{TS} \right\rceil (l_i + \xi) + C_i$$

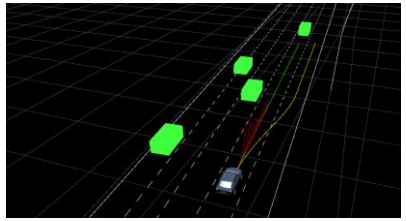


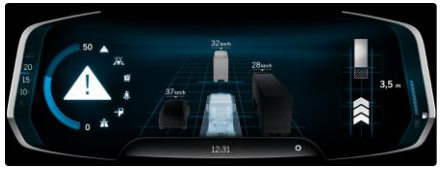
- TS and 3 priority/interleaving levels might not be enough...
- Given (P,B,D) how to compute (TS, priority)? **NOT TRIVIAL**

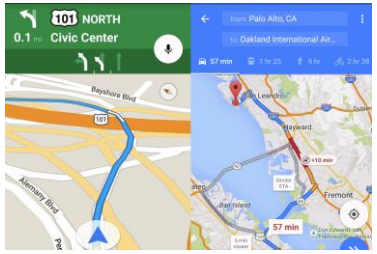
Our prototype EDF+CBS on GPU

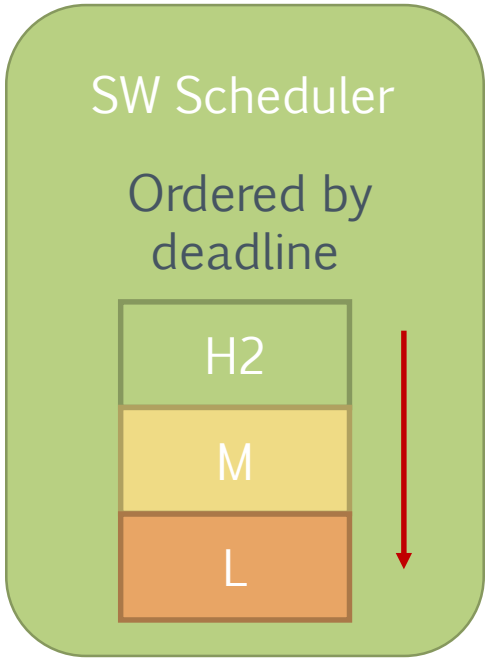


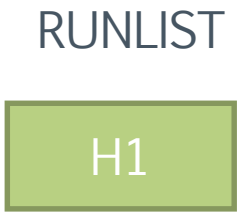
Our prototype EDF+CBS on GPU

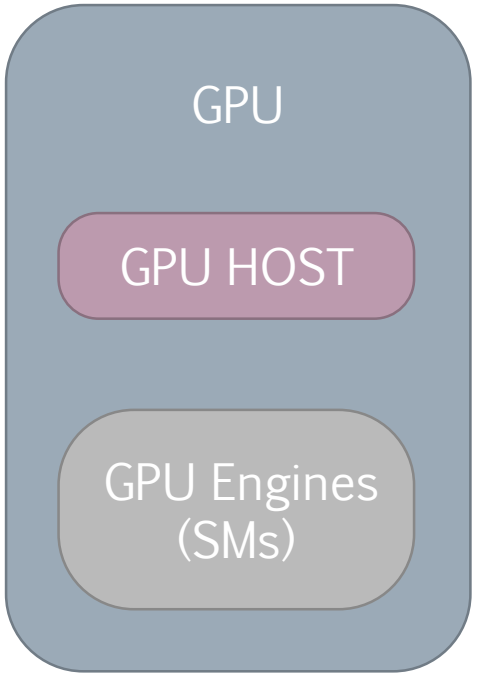












APIs extension for scheduling parameters: batch of cmds

› We need to define a deadline **granularity**:

- Whole program: not enough flexibility
- Single commands/Kernel/Shader invocations: too fined graned!
- **Batch of commands:** allows us to define task boundaries among many mem. ops. and kernel/shader invocations

Problem: we need to define API extension to group commands into batches

- **OpenGL**: Cmds are already batched per frame submission. We set a desired framerate target and a worst case rendering time estimation.
- **CUDA**: Exploit CUDA Streams and we prototype CUDA specific runtime calls for command batching and submission of scheduling parameters.

CUDA and OpenGL API extension

OpenGL

```
init_function(){
  //Load data, geometries, textures
  //And compile shaders...
  glSetFrameTarget(framerate, budget_us);
}
Render_loop(){
  //uniforms and attributes updates
  //drawcalls so on...
  glSwapBuffers();
  //Kicks to GPU and waits as specified in
  //init function.
}
```

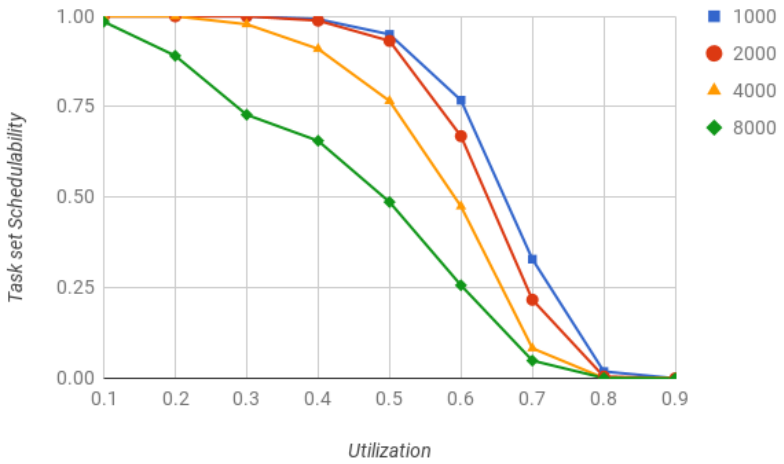
CUDA

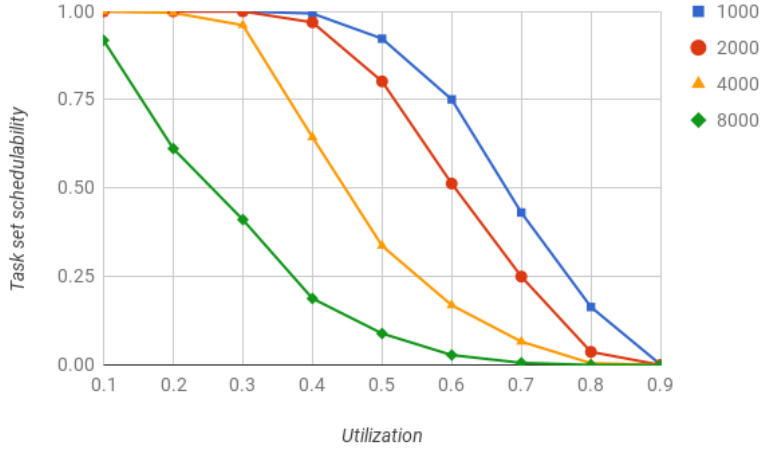
```
//CUDA ctx creation and data initialization
 init_function(){
   cudaStream_t s0, s1, ..., sn;
   cudaStreamCreate(&s0); ...
   cudaDeviceSynchronize();
 }
//CPU thread0
while (wait_for_new_data()){
cudaStreamDeadlineBegin(s0,Dr0,B0,P0);
//cuda kernels, memcpy etc... on stream s0
cudaStreamDeadlineEnd(s0);
}
```

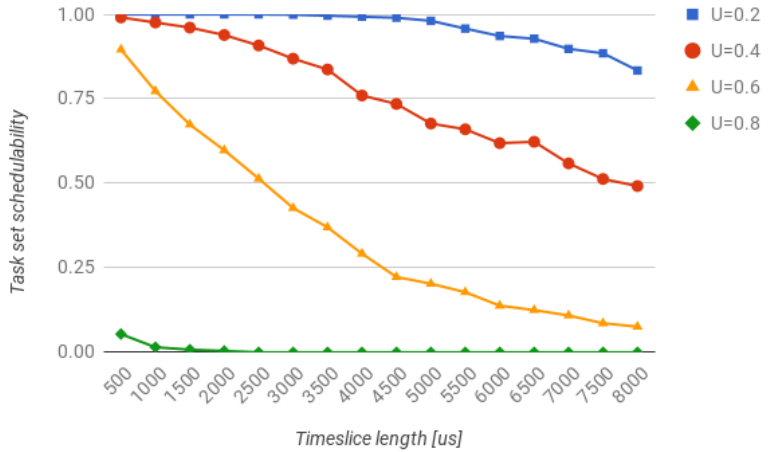
Test and Experiments

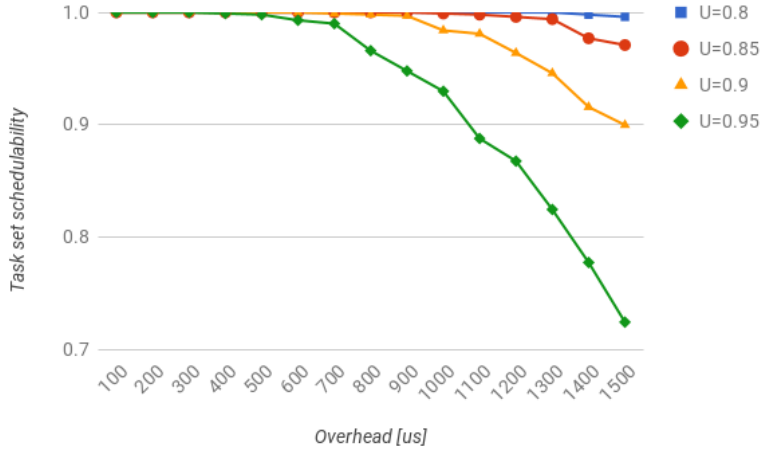
- Simulated benchmarks:
  - UUnifast algorithm for task generation
    - P : [16, 125] ms
    - U : [0.1, 0.9] (for Real Time tasks)
    - D = P
    - Only one best effort task that continuously submits work (total U > 1)
- On board experiments [NVIDIA Drive PX2 Autocruise]
  - OpenGL graphic app (30 FPS, RT task)
  - CUDA 10-layer CNN (P=40ms, B=3ms, D=4ms, RT task)
  - Two best effort graphic tasks (ctree, gears)

Simulated benchmarks

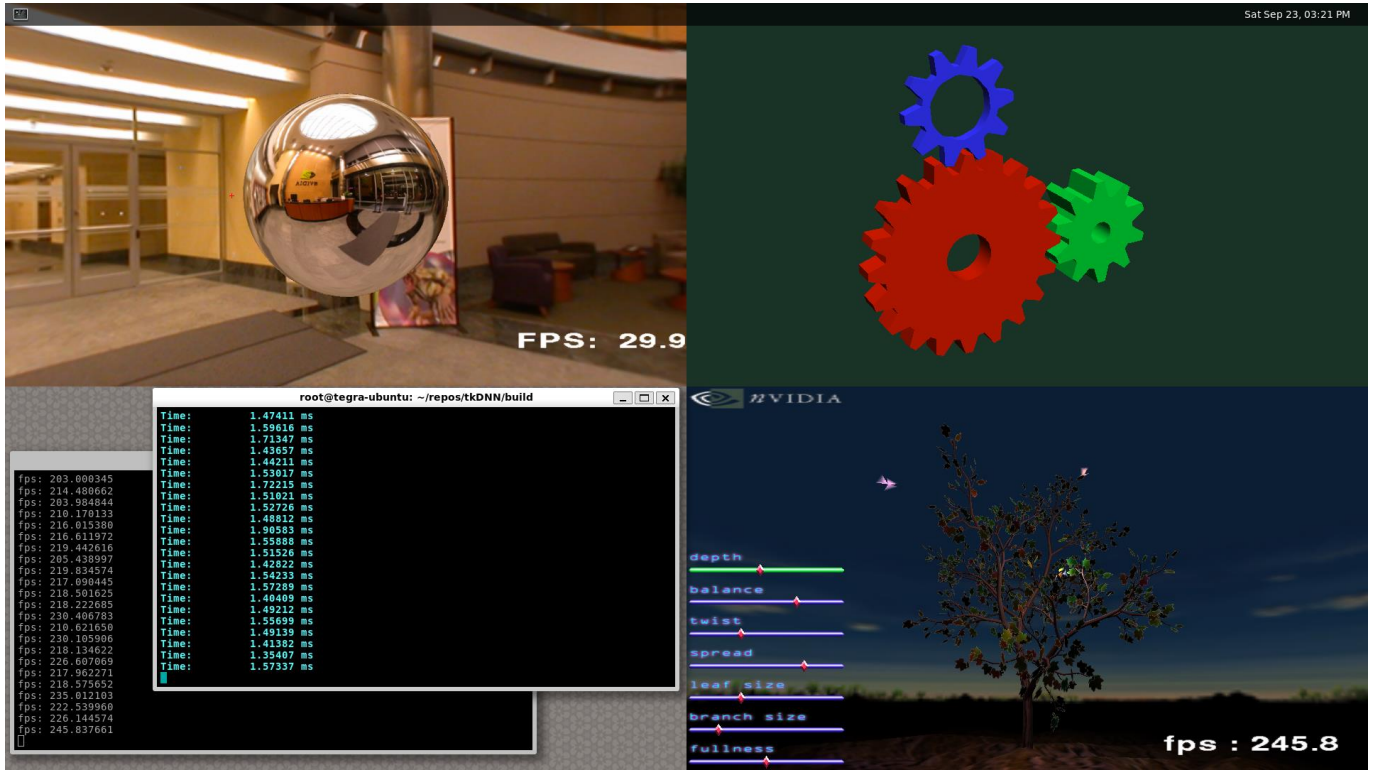








On Board experiments



Task class	WCRT decrease [Interleaved vs EDF %]	Deadline miss [Interleaved % | EDF %]
RT CUDA CNN	64.56%	0.55% | 0%
RT Graphic app	93.35%	4.02% | 0%
BE Graphic app (60 FPS)	17.70%	-
BE Graphic (unbounded)	(increase) 158.84%	-

Conclusion

- Working prototype of an EDF+CBS scheduler for GPUs
- Disclosed NVIDIA Native GPU application scheduler
- Can be ported to run native (no Hypervisor)
- In the future:
  - Limited preemption (Bertogna et al.)
  - GRUB (Lipari et al.)
  - More complex task models for more complex boards (NV Xavier: GPU, DLA, PVA …)

ACKNOWLEDGMENT
The authors would like to thank NVIDIA Corporation for letting us work on their GPU architectures and for the useful information given to us. This work is also part of the **Hercules** and **OPEN-NEXT** projects, which are respectively funded by the EU Commission under the HORIZON 2020 framework program (GA-688860) and ERDF-OP CUP E32I16000040007.