The case for limited-preemptive scheduling in GPUs for real-time systems

Roy Spliet, Robert Mullins
Department of Computer Science and Technology, University of Cambridge

***Abstract*—Many emerging cyber-physical systems, such as autonomous vehicles, have both extreme computation and hard latency requirements. GPUs are being touted as the ideal platform for such applications due to their highly parallel organisation. Unfortunately, while offering the necessary performance, GPUs are currently designed to maximise throughput and fail to offer the necessary hard real-time (HRT) guarantees.**

In this work we discuss three additions to GPUs that enable them to better meet real-time constraints. Firstly, we provide a quantitative argument for exposing the non-preemptive GPU scheduler to software. We show that current GPUs perform hardware context switches for non-preemptive scheduling in 20-26.5μs on average, while swapping out 60-270KiB of state. Although high, these overheads do not forbid non-preemptive HRT scheduling of real-time task sets. Secondly, we argue that limited-preemption support can deliver large benefits in schedulability with very minor impact on the context switching overhead. Finally, we demonstrate the need for a more predictable DRAM request arbiter to reduce interference caused by processes running on the GPU in parallel.

I. Introduction

An important class of cyber-physical systems now demand both significant compute and hard real-time (HRT) support. A prime example is the autonomous vehicle, where low-latency engine control systems are combined with time-critical AI classification and decision making procedures. These classification problems are solved using massively parallel algorithms such as neural networks [17]. From both a cost and performance-per-watt perspective it is attractive to offload these problems to massively parallel accelerators like GPUs. NVIDIA's introduction of Drive PX computers for assisted- and autonomous driving [2] are evidence of the shift of GPUs towards the domain of safety-critical HRT systems.

GPUs are designed with some real-time principles in mind, for example to resolve contention for the DRAM bus such that it never leads to a flickering image. Unfortunately, these real-time provisions are not applicable to the emerging cyber-physical use-cases. Instead, there is a strong desire to bound execution- and response time of GPU compute workloads.

A prerequisite to bound the worst-case response time of HRT tasks and to determine schedulability is a thorough understanding of the task scheduling policy. NVIDIA GPUs allow FIFO scheduling of kernels plus limited (sparsely documented) support for prioritising some kernels over others within the same hardware context [6]. Additionally, hardware supports non-preemptive context-switching between processes as a means to provide security mechanisms like per-task virtual memory spaces. Unfortunately, the criteria used for scheduling processes are unknown and not under the control of system developers. To achieve HRT scheduling on GPUs, systems must instead introduce a software abstraction layer [11], [14], [15], [16], [22]. These systems add overhead and force all tasks in a single hardware context, sacrificing inter-task protection mechanisms. Furthermore, their non-preemptive scheduling still imposes large worst-case blocking times on task sets, reducing HRT schedulability.

In this paper we present a case for three changes in GPU architecture. Firstly, we argue that exposing control over the current non-preemptive GPU context switching mechanisms to systems developers can facilitate low-overhead HRT task scheduling while providing desirable security mechanisms. From measurements we observe an average context switching time on NVIDIA GPUs of $20-26.5\mu$s. Using these numbers, we demonstrate the schedulability properties of random task sets under overhead-aware non-preemptive earliest-deadline first (npEDF) scheduling. Secondly, we motivate from an HRT perspective the proposal of Tanasic et al. [24] to perform context switches on the boundary of a work-group (SM draining) rather than the compute kernel. Measured by the schedulability of randomly generated task sets under limited-preemptive EDF scheduling, we show that this solution provides a good trade-off between blocking time and context switching overhead. By contrast, we show that fully preemptive scheduling on GPUs will perform similar or worse than non-preemptive scheduling as a result of high expected context switch overheads when exposed to task sets of the same parameters. Finally, we show the need for a predictable and analysable DRAM subsystem to provide optimistic bounds on the latency of GPU compute workloads. Using our measurement set-up, we expose interference between display scan-out and context switching by showing that increasing the bandwidth demand of scan-out increases the worst-case context switch time from $3.7\times$ average to more than $5.5\times$.

II. Background and related work

A. GPU nomenclature

Developers implement their data-parallel algorithms in one or more *compute kernels*, following the *Single Program, Multiple Data streams* (SPMD) programming model. A kernel typically describes the transformations on a single data element in the data stream. Hardware will spawn one thread or *work-item* for every data element.

Following OpenCL nomenclature, work-items are grouped into *work-groups*. On NVIDIA hardware a work-group consist

of multiple 32-thread groups called *warps* (AMD: *wavefronts*). Each warp will typically be executed in a SIMD fashion.

To execute SPMD programs, NVIDIA hardware implements the *Single Instruction, Multiple Threads* (SIMT) execution model [18] following a hierarchical structure. At the bottom level, a *Streaming Multiprocessor* (SM) contains many computational *cores* on which work is dispatched by *warp schedulers*. A warp scheduler issues one or two SIMD instructions per clock cycle at a warp granularity, temporally interleaving the instructions of multiple warps to minimise hardware stalls. A large register file ensures that the warp scheduler can interleave instructions of warps from the same hardware context with zero overhead. Further up the hierarchy, one or more SMs are contained within a *Graphics Processor Cluster* (GPC). A GPU contains one or more GPCs.

B. Non-preemptive context switching on NVIDIA GPUs

On current NVIDIA hardware a context switch is performed by dedicated custom "Falcon" microcontrollers [28], [29]: one at the top-level called *FECS* (Front-End Context Switch) and one per GPC called *GPCCS* (GPC Context Switch). Each microcontroller is connected to a set of FIFO buffers, used to coalesce register read/write actions to memory to improve DRAM efficiency. At the top-level, a hardware scheduling unit triggers context switches by notifying FECS.

When FECS receives a context switch request, it configures all execution engines (SMs, rasterisers etc.) to pause after finishing the currently running compute kernel. Once engines are paused, it notifies each GPCCS to swap state. FECS and GPCCS microcontrollers proceed by writing the MMIO address of every register that must be saved to their FIFOs. After all FIFOs are drained and their register values stored, the reverse process is initiated to restore registers of the next context. Finally the GPCCSs signal completion, after which FECS resumes execution of all engines.

Tanasic et al. [24] explore implementations for full- and limited-preemptive context switching on NVIDIA GPUs. They evaluate their approach using an in-house simulator by measuring average context switching times for several benchmarks. We extend this work by presenting a baseline for context switching under non-preemptive scheduling (henceforth "non-preemptive context switching") on commodity hardware and evaluating preemption models from an HRT point of view.

C. Real-time considerations for GPUs

We consider three key differences between popular GPU architectures and the CPU: the SIMT execution model, the lack of direct I/O access to external devices from GPU compute cores, and the absence of shared memory resources between different compute kernels.

SIMT execution allows GPUs to achieve high resource utilisation by executing the many work-items of a compute kernel on all available GPCs in parallel. We limit ourselves to the base case of temporal multitasking, in which case GPUs are best analysed as a *uniprocessor* where each compute kernel represents a task in the system. Limited support exists for spatial multitasking of kernels within a context [6], but in the absence of scheduler implementation details we consider this a throughput optimisation without analysable worst-case response time benefits.

NVIDIA GPUs will never encounter context switches due to self-suspending jobs. In traditional systems we can categorise self-suspensions in three classes: jobs waiting to be granted access to a shared resource, jobs blocked on I/O and jobs explicitly yielding their core. Alglave et al. [5] show that sharing resources between different jobs on a GPU is deemed infeasible by the weak memory consistency model found on current GPUs. I/O blocking is impossible because the GPU is a slave device without direct access to external devices. Finally, NVIDIA GPUs do not support a yield instruction. As a consequence of not encountering self-suspension we can bound the number of context switches in a system.

D. System model

In this work we consider the *periodic task model* [19] with *implicit deadlines*. For limited-preemptive execution, this model defines a set of tasks τ of size n where each task τ_i is described by a three-tuple (c_i, p_i, q_i). During execution, each task releases a series of *jobs* $J_{i,k}$. The *period* p_i describes the time between two successive job releases from the same task. In an implicit deadline system, a job's absolute deadline equals its launch time plus p_i. The *cost* c_i is the worst-case execution time (WCET) of a job. The final parameter q_i describes the maximum preemption delay or "non-preemptive blocking period". The utilisation of a task $U_i = c_i/p_i$ and the utilisation of a task set $U_\tau = \sum_{1 \leq i \leq n} U_i$.

We limit our experiments to EDF scheduling [19]. Although not implemented by commodity GPUs, EDF's optimality among both preemptive- and non-preemptive non-idling uniprocessor schedulers [12] removes a factor of uncertainty from the cause of a task set's non-schedulability. This results in a more accurate demonstration of the influence of the context switch times in our experiments.

Two concepts underlie EDF schedulability analysis. Firstly, the *critical instant* is the instant for which a task's response time is maximised [19]. For preemptive EDF this instant corresponds with the *synchronous arrival sequence*, releasing the first job of each task at time $t = 0$ and each subsequent job $J_{i,k}$ at time $t = k * p_i$. Secondly, Baruah et al. [7] define the concept of *demand bound* as the sum of the cost of all *jobs* in the critical instant whose absolute deadline is on or before t. We define $h(\tau_i, t)$ as the function returning this bound for task τ_i.

Building on this work, Baruah [8] proved that under EDF scheduling, limited preemptive (implicit deadline) task sets are not schedulable iff:

$$\exists t : 0 \leq t : \sum_{i=1}^{n} h(\tau_i, t) > t$$

or there is a $\tau_j, 1 \leq j \leq n$, and

$$\exists t : 0 \leq t < p_j : q_j + \sum_{i=1, i \neq j}^{n} h(\tau_i, t) > t$$

NVIDIA GeForce	SM #	GPC MHz	DRAM GiB/s	State KiB	Measured time (μs) Min	Avg	Max	Avg. BW util GiB/s	%
GT 710	1	953	14.4	63.9	9.2	21.5	80.1	2.83	19.6%
GT 640	2	901	28.5	68.2	13.6	26.5	43.7	2.45	8.6%
GTX 650	2	1058	80.0	68.2	12.7	23.2	36.0	2.71	3.4%
GTX 780	12	992	288.4	268.6	9.7	20.0	28.6	13.76	4.8%

TABLE I
MEASURED CONTEXT SIZE AND SWITCHING OVERHEAD

For schedulability analysis of non-preemptive tasks, we can define $\forall i \in [1,n], q_i = c_i - 1$, resulting in the original npEDF schedulability conditions ([12], [13]).

To date, the best algorithm to bound the set of relevant values for t is Zhang et al's QPA [30]. Short's [23] *lpQPA-LL* extends QPA with schedulability analysis of implicit-deadline periodic task sets under limited-preemptive EDF.

Under EDF scheduling, the number of context switches is upper bound by two per job [10]. The rationale is that a reactive implementation of this policy only takes decisions on two types of events: job release and job completion. For non-preemptive EDF, scheduling decisions caused by job releases are postponed until after completion of the current job. This tightens the upper bound to one context switch per job.

III. CONTEXT SWITCHING OVERHEAD

To make substantiated claims about the effectiveness of preemption models for GPUs, in this section we present the results of measuring context size and switching time on NVIDIA GPUs. By manipulating measurement conditions, we also demonstrate the effect of performance interference on worst-case context switch times, motivating further research in predictable DRAM subsystems for GPUs.

A. Measurement set-up

In this experiment we measure the size and switching time of non-preemptive contexts on several NVIDIA Kepler generation (2012-2014) graphics cards. Measurement is performed by a modified context switching firmware. The nature of our changes mandate the use of the open source "nouveau" driver for NVIDIA graphics cards [1] rather than the official driver. Source code and acquired data is available at https://github.com/RSpliet/RTGPU-Preempt.

We modify the FECS firmware to report context switching time in an available scratch register. This time spans from the moment all GPCs are paused to the moment they resume. We measure the context size and switching times using an instrumentation tool built using the envytools suite.

The timer used for this measurement has a granularity of 32ns. Our firmware modifications increase the runtime of a context switch by two register read operations. Based on 1,064,960 samples we determine that these operations skew our measurement by 160-224ns, averaging at 176ns.

GPUs are connected to a monitor operating at 1600x1200@60Hz. To trigger context switches, we run two generic workloads in separate contexts (XFCE on Xorg, windowed OpenArena @1024x768). The choice of workload should have minimal effect on the measured overheads, as all SMs are paused during the measured interval. We use our instrumentation tool to obtain 20 million samples per GPU.

B. Results

The fifth column in Table I lists the size of the state that needs to be stored to memory on a non-preemptive context switch. This state, significantly larger than that of a modern CPU, includes OpenGL/CUDA/OpenCL configuration, hardware settings, a pointer to the top-level page-table, and many other undocumented pieces of information. The contents of the register- and local-memory file are not included.

Such large state results in observed context switch times in the order of tens of microseconds. Our measured average context switch time (column 7) corresponds with NVIDIA's claim [26] of $\sim$25μs for the Fermi-generation of graphics cards (2010-2012). Such overhead clearly needs to be accounted for when performing schedulability analysis.

Experiments with lower GPC clocks, leaving all other clocks (including the DRAM interface) unaltered, reveals that average context switch time increases. This suggests that the process of context switching is not solely memory bound. However, the observed worst-case context switch times on the low-end GeForce GT710 are slightly lower ($<$5%) when the GPC clock is reduced by 15%. This worst case overhead reduction rules out the theory that context switch is compute bound in the worst case. Instead, data indicates that higher worst-case context switch times correlate with lower DRAM bandwidth. We will present further evidence of context switching being memory bound in the worst case in Section III-C.

Figure 1 shows a logarithmic histogram of samples for the GeForce GT710, displaying the extent to which our maximum sample introduces pessimism to schedulability analysis. We observe that the vast majority of the samples lie around the average of 21.5μs, whereas merely $\sim$0.3% of the samples lie in the tail of the measurement. The observed maximum is $\sim$3.7$\times$ average.

In the next section we demonstrate how interference affects the samples in this tail. In the light of these results we discuss the limitation of empirical measurements.

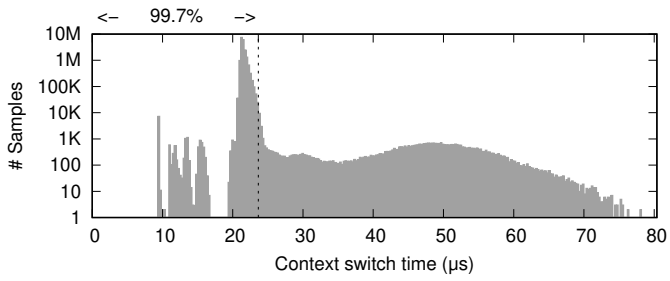

Fig. 1. Histogram of context switching overhead on NVIDIA GeForce GT710

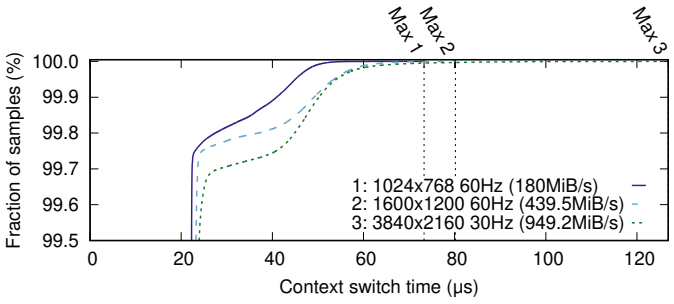

Fig. 2. Cumulative histogram of context switching overhead on GeForce GT710

C. Interference effects

To demonstrate interference within a GPU we repeat the experiment from Section III-B with different display resolutions. Figure 2 shows a cumulative histogram displaying the top 0.5% samples of context switch times of this experiment. From this graph we observe that increasing the required bandwidth for scan-out has a strong negative effect on the observed worst-case context switching overhead.

This interference is caused by sharing the DRAM subsystem between multiple workloads. If we consider a DRAM hierarchy, we find one or more channels on the top level. Each channel has a data bus to its RAM chips. If two memory operations transfer data from/to the same channel, these requests need to be serialised by an arbiter. This arbiter implements a prioritisation policy that makes a trade-off between performance and latency. If this policy is predictable it could be possible to determine a worst-case latency on individual memory requests, but unfortunately the prioritisation policy of GPU memory controllers is unknown.

Scan-out is merely one example of a GPU subsystem that requires access to DRAM in parallel with context switching. Other examples include DMA transfers and video decoding. Indeed, our observations on interference give reason to believe that e.g. the proposal of Verner et al. [25] to overlap DMA transfers with execution is likely to decrease response time predictability unless measures are taken to account for DRAM interference. Without analysable architectures and models, it is impossible to use quantitative measurements like these to distinguish between the worst-case execution time of a workload and its worst-case response time. This results in pessimistic GPU timing analysis.

In the next section we show how measured and extrapolated context switch times affect schedulability. We use these results to motivate further research in GPU preemption models.

IV. Schedulability analysis

To illustrate the effects of context switching overheads on schedulability, we performed a schedulability analysis, comparing non-preemptive, limited-preemptive and full-preemptive EDF. Next we explain how these scheduling policies map to microarchitectural solutions.

A. Models

Based on measured context switch overheads for non-preemptive execution on NVIDIA GeForce GT640, similar in specifications to the embedded Tegra K1 SoC, we extrapolate parameters for EDF and lpEDF. Resulting estimates are summarised in Table II.

For these estimates we make two simplifying assumptions. Firstly, we disregard cache-related preemption delays as they depend too much on the application and GPU microarchitecture to allow substantial claims. Secondly, divergence between warp schedulers will cause some SMs to wait idle for the last to finish. This idle time negatively affects the WCET of jobs. However, without knowledge of the scheduling policy implemented within the warp-schedulers, we cannot determine a bound on the divergence of warps in flight. This prevents us from modelling this effect in our analysis.

Scheduler policy	State (KiB) Ctx	Reg	Local	Total	Time (μs) Avg	Max	Preempt /job [10]
EDF	68.2	512	96	676.2	263	434	$\times 2$
lpEDF	68.2	0	0	68.2	27	44	$\times 2$
npEDF	68.2	0	0	68.2	27	44	$\times 1$

TABLE II
Parameters for schedulability analysis

Non-preemptive scheduling is currently implemented on NVIDIA GPUs. For non-preemptive EDF analysis we inflate c_i with the measured cost of one context switch.

Limited-preemptive scheduling applies to *SM draining* [24], a hardware solution that allows compute kernels to be preempted on the boundary of a work-group. At these boundaries the register and local memory contents do not need to be preserved, hence the state size and context switch time is estimated equal to that of the non-preemptive case. We account for this by inflating each task's cost by $2\times$ the measured context switching overhead.

For (full-)preemptive scheduling we must account for the larger context required to preserve register and local memory contents. To estimate the context switch time, we assume a linear correlation with the context size. Despite evidence that the DRAM subsystem provides more efficiency for bigger transfers on average [24], we cannot make optimistic assumptions for the worst-case without further research. For preemptive scheduling we inflate each task's cost with $2\times$ the projected context switching overhead.

B. Measurement set-up

For each utilisation $U \in (0.2, 0.21..1.0)$, we generated 100,000 implicit-deadline periodic task sets. Tasks have a period between 1,000 and 15,000 (μs), modelling kernels across the range of costs observed by Tanasic et al. [24]. Utilisation is randomly assigned to each task with a uniform distribution using the UUniFast algorithm [9].

Schedulability tests are performed using Brandenburg et al's schedcat, modified to support lpQPA-LL. [23] For limited preemption, we set $q_i = c_i / rand(5, 500)$, corresponding with $5 - 500$ work-groups per SM.

C. Schedulability

Figure 3 shows the result of this schedulability experiment when generating task sets of two tasks. We draw two conclusions from this graph. Firstly, the large overheads we measured do not prevent HRT schedulability. However, the large projected overhead greatly reduces the value of full-preemptive scheduling. Assuming worst-case context switch times, we find a minimal benefit for task sets with $U_\tau \leqslant 0.71$. For full-preemptive scheduling to become feasible in a real-time GPU, the overhead must ideally be bound to a value close to the average projection.

Secondly, we see that a limited-preemptive scheduler can benefit from the combination of paying the context switching overhead of non-preemptive scheduling and achieving response times close to preemptive scheduling. In practice this

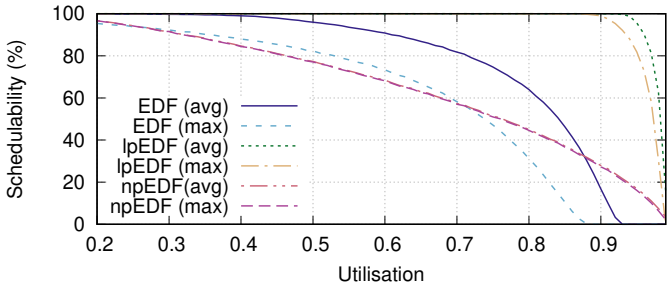

Fig. 3. Schedulability: 2 tasks, $c_i \in [1000, 15000]\mu$s

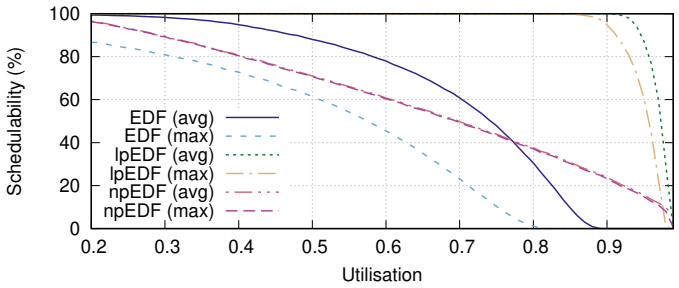

Fig. 4. Schedulability: 3 tasks, $c_i \in [1000, 15000]\mu$s

means that for the chosen parameters, we can schedule 99% of all task sets with $U_\tau \leqslant 0.89$ even when the worst case context switching time is assumed.

To show the influence of the task set size on schedulability, Figure 4 shows the results of this experiment when generating task sets of size 3. For preemptive-scheduling, the maximum projected overhead now outweighs the theoretical benefits of preemption completely. However, under limited-preemption EDF we would continue to be able to schedule many high-utilisation task sets.

D. Maximum blocking exploration

To demonstrate the impact of the maximum blocking parameter q, we perform an lpEDF schedulability analysis on random task sets where for a work-groups/SM ratio $w \in [2, 30]$, $q_i = c_i/w$. Figures 5 and 6 show the results of this analysis with w on the x-axis. On the y-axis we find the maximum utilisation for which $> 90\%$ (Figure 5) and $> 99\%$ (Figure 6) of the task sets are schedulable. We generated task sets with 3 tasks, for each task $c_i \in [1000, 15000]\mu$s

We see that even for two work-groups/SM, 99% of all task sets with $U_\tau < 0.29$ are schedulable. In Figure 4 we observe that this outperforms non-preemptive scheduling. Furthermore, for jobs containing 9 or more work-groups/SM, the figures demonstrate that the deciding factor for schedulability is not the preemption delay but rather the context switching overhead. For reference, 9 work-groups/SM corresponds to 122,880 work-items (e.g. a 351×351 image or matrix) on the largest Kepler generation GPU, the NVIDIA GeForce GTX780 TI. Such data sets are realistic for AI and computer vision workloads, supporting our claim that lpEDF kernel scheduling will result in increased GPU utilisation under HRT constraints.

V. Discussion and Future Work

A. DRAM interference

In Section III-C we explore the interference between context switches and display scan-out to show how contention for the DRAM subsystem reduces response time predictability. However, interference does not solely occur between these two tasks. Concurrent DMA- or video decoding activity further diminishes the worst-case latency of individual requests.

There are two known ways to mitigate this interference. Firstly, DRAM partitioning (e.g. bank privatisation [21]) could be applied to isolate subsystems. Although this has far-reaching consequences to the freedom a system has to allocate memory to each workload, it could serve as a way to reduce the worst-case interference on commodity hardware.

Secondly, designing a real-time DRAM request arbiter that prioritises requests based on their time of arrival and/or criticality level could make this interference predictable and analysable. Such arbiters have been studied for traditional multi-core architectures connected to DDR2 and DDR3 memory (e.g. [4], [20]), and prove effective at bounding the response time of individual request. Unfortunately, as improvements in DRAM latencies continue to stagnate and data buses are becoming wider, the bandwidth utilisation of memory controllers with such arbiters gets successively worse with each DRAM generation [27]. Future research should explore the design space of bound-latency high-throughput DRAM subsystems for GPUs under the constraints of present-day DRAM.

B. Task scheduling

In Section IV we describe how the limited-preemption model is a good fit for GPUs, assuming it reduces the maximum blocking time at a cost similar to that of non-preemptive context switching. One reason why this assumption could be too optimistic is that it disregards the context of subsystems that are irrelevant for most compute workloads, e.g. the rasteriser. The rasteriser keeps track of a lot of state during execution and does not appear to work on the granularity of work-groups. This raises questions on how the state of such fixed-function components should be treated in the preemptive execution models: Is it desirable to perform a context switch on these components in lock-step with the

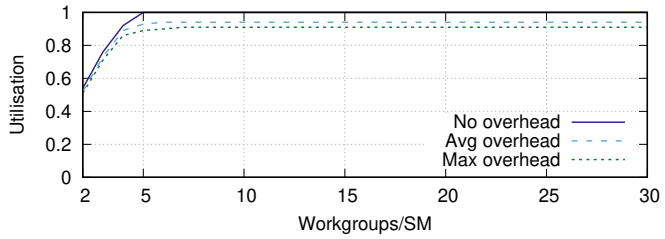

Fig. 5. Impact of work-groups/SM on 90% schedulability

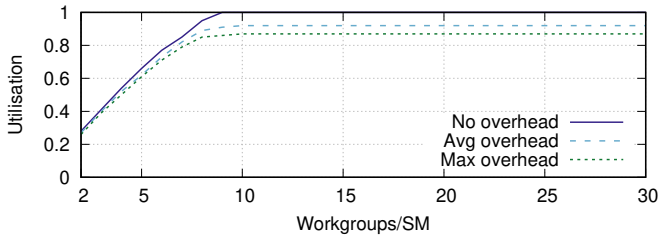

Fig. 6. Impact of work-groups/SM on 99% schedulability

compute subsystem? Can we find points in time at which these components have less state? Do we need to take the state of these components into account for (non-rendering) real-time compute workloads, or is it possible to take this out of the equation? Would this require the design of new, compute-oriented architectures?

Another avenue for research is the concept of GPU partitioning or "spatial multitasking" [3]. A partitioned non-preemptive GPU could permit a cost-based grouping of tasks, providing lower-latency guarantees to shorter tasks. Research could determine the architectural overhead of GPU partitioning, the implications to the context size, the schedulability implications for HRT workloads and the implications of DRAM-related interference on response-time analysis.

VI. Conclusion

In this work we have motivated the need for research in three areas of GPU design for real-time applications. Firstly, we show that it is possible to use existing non-preemptive EDF schedulability analysis to prove schedulability of task sets under the parameters we expect for massively parallel applications in the HRT domain running on contemporary GPUs. Prerequisite is that GPUs provide control over their non-preemptive task scheduler to software. We show that the measured average context switching overhead of 20-26.5μs has only a limited influence on schedulability. Secondly, we motivate research in limited-preemptive scheduling following the "SM draining" approach [24] to reduce the maximum blocking time of tasks while retaining the security benefits of task isolation. We show that this can result in significantly higher schedulability of task sets. Finally, we show that interference effects caused by contention for the shared DRAM subsystem has a negative effect on observed worst-case execution times of individual tasks. We suggest that further research should be conducted towards bound-latency DRAM request arbiters that enable more optimistic worst-case response times with minimal sacrifices to throughput.

Acknowledgements

We thank Andy Ritger (NVIDIA) and Joonas Lahtinen (Intel OTC) for their technical discussion, and Timothy Jones (University of Cambridge, Dept. CST) for his feedback.

References

[1] Nouveau: Accelerated Open Source driver for NVIDIA cards. URL: https://nouveau.freedesktop.org/wiki/.

[2] NVIDIA Tegra X1 - NVIDIA's New Mobile Superchip, 2015.

[3] J. T. Adriaens, K. Compton, N. S. Kim, and M. J. Schulte. The case for GPGPU spatial multitasking. In *IEEE Int. Symp. on High-Performance Comp. Arch.*, pages 1–12, Feb 2012.

[4] B. Akesson, K. Goossens, and M. Ringhofer. Predator: A Predictable SDRAM Memory Controller. In *Proc. of the 5th IEEE/ACM Int. Conf. on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '07, pages 251–256. ACM, 2007.

[5] J. Alglave, M. Batty, A. F. Donaldson, G. Gopalakrishnan, J. Ketema, D. Poetzl, T. Sorensen, and J. Wickerson. GPU Concurrency: Weak Behaviours and Programming Assumptions. *SIGPLAN Not.*, 50(4):577–591, March 2015.

[6] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith. GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed. In *Proc. 38th Real-Time Systems Symp.*, pages 104–115, Dec 2017.

[7] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proc. 11th Real-Time Systems Symp.*, pages 182–190, Dec 1990.

[8] Sanjoy Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *17th Euromicro Conf. on Real-Time Systems*, pages 137–144, July 2005.

[9] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[10] A. Burns, K. Tindell, and A. Wellings. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Trans. on Software Engineering*, 21(5):475–480, May 1995.

[11] G.A. Elliott, B.C. Ward, and J.H. Anderson. GPUSync: A Framework for Real-Time GPU Management. In *Proc. 34th Real-Time Systems Symp.*, pages 33–44, Dec 2013.

[12] L. George, P. Muhlethaler, and N. Rivierre. Optimality and non-preemptive real-time scheduling revisited. Research Report RR-2516, INRIA, 1995. Projet REFLECS. URL: https://hal.inria.fr/inria-00074162.

[13] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of period and sporadic tasks. In *Proc. 12th Real-Time Systems Symp*, pages 129–139, Dec 1991.

[14] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar. RGEM: A Responsive GPGPU Execution Model for Runtime Engines. In *Proc. 32nd Real-Time Systems Symp.*, pages 57–66, Nov 2011.

[15] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa. TimeGraph: GPU scheduling for real-time multi-tasking environments. In *USENIX ATC11*, page 17, 2011.

[16] S. Kato, M. McThrow, C. Maltzahn, and S. Brandt. Gdev: First-class GPU resource management in the operating system. In *USENIX ATC12*, volume 12, 2012.

[17] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada. An Open Approach to Autonomous Vehicles. *Micro, IEEE*, 35(6):60–68, Nov 2015.

[18] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro*, 28(2):39–55, March 2008.

[19] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20(1):46–61, January 1973.

[20] M. Paolieri, E. Quiñones, and F. J. Cazorla. Timing Effects of DDR Memory Systems in Hard Real-time Multicore Architectures: Issues and Solutions. *ACM Trans. Embed. Comput. Syst.*, 12(1s):64:1–26, Mar 2013.

[21] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee. PRET DRAM controller: Bank privatization for predictability and temporal isolation. In *Proc. of the 9th IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis*, pages 99–108, Oct 2011.

[22] C. J. Rossbach, J. Currey, M. Silberstein, B. Ray, and E. Witchel. PTask: Operating System Abstractions to Manage GPUs As Compute Devices. In *Proc. of the 23rd ACM Symp. on Operating Systems Principles*, SOSP '11, pages 233–248. ACM, 2011.

[23] M. Short. Improved schedulability analysis of implicit deadline tasks under limited preemption EDF scheduling. In *ETFA2011*, pages 1–8, Sept 2011.

[24] I. Tanasic, I. Gelado, J. Cabezas, A. Ramirez, N. Navarro, and M. Valero. Enabling preemptive multiprogramming on GPUs. In *ACM/IEEE 41st Int. Symp. on Computer Architecture (ISCA)*, pages 193–204, 2014.

[25] U. Verner, A. Schuster, and M. Silberstein. Processing Data Streams with Hard Real-time Constraints on Heterogeneous Systems. In *Proc. of the Int. Conf. on Supercomputing*, pages 120–129, 2011.

[26] C.M. Wittenbrink, E. Kilgariff, and A. Prabhu. Fermi GF100 GPU Architecture. *Micro, IEEE*, 31(2):50 –59, March-April 2011.

[27] Z. P. Wu, Y. Krish, and R. Pellizzoni. Worst Case Analysis of DRAM Latency in Multi-requestor Systems. In *Proc. 34th Real-Time Systems Symp.*, pages 372–383, Dec 2013.

[28] J. Xie. NVIDIA RISC-V Evaluation Story. 4th RISC-V Workshop, 2016. URL: https://www.youtube.com/watch?v=gg1lISJfJI0.

[29] Y.Fujii, T. Azumi, N. Nishio, and S. Kato. Exploring Microcontrollers in GPUs. In *Proc. 4th Asia-Pacific Workshop on Systems*, pages 2:1–2:6. ACM, 2013.

[30] F. Zhang and A. Burns. Schedulability Analysis for Real-Time Systems with EDF Scheduling. *IEEE Trans. on Computers*, 58(9):1250–1258, Sept 2009.