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

Roy Spliet  Robert Mullins (first.last@cst.cam.ac.uk)

Department of Computer Science and Technology
University of Cambridge
GPUs in real-time systems

Work complementary to Amert et al.\(^1\):

- Prior work: scheduling tasks within single context (mainly).
- This work: scheduling properties of different HW contexts.

\(^1\)Amert, T., Otterenes, N., Anderson, J.H., Smith, F. D., GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed, December 2017
Outline

Background: context switch mechanisms and preemption models

Experiment: Measure context switch response times

Experiment: Task scheduling on GPUs

Conclusion
Context switch mechanisms and preemption models

Kernel invocation

clProcessImage: 1920 * 1080 = 2,073,600 threads

\(^2\)Tanasic et al. “Enabling preemptive multiprogramming on GPUs, 2014”
Non-preemptive scheduling (current GPUs):

- Finish whole kernel.
- Max blocking: WCRT of kernel.
- Swap: HW+OpenCL configuration.

Tanasic et al. “Enabling preemptive multiprogramming on GPUs, 2014”
Context switch mechanisms and preemption models

Non-preemptive scheduling (current GPUs):
- Finish whole kernel.
- Max blocking: WCRT of kernel.
- Swap: HW+OpenCL configuration.

Preemptive scheduling\(^2\):
- Interrupt anywhere.
- Max blocking: none.
- Swap: HW+OpenCL configuration, register files, local memory.

\(^2\)Tanasic et al. “Enabling preemptive multiprogramming on GPUs, 2014”
Context switch mechanisms and preemption models

Non-preemptive scheduling (current GPUs):
- Finish whole kernel.
- Max blocking: WCRT of kernel.
- Swap: HW+OpenCL configuration.

Preemptive scheduling:
- Interrupt anywhere.
- Max blocking: none.
- Swap: HW+OpenCL configuration, register files, local memory.

Limited-preemptive scheduling:
- Interrupt on work-group boundary ("SM draining")\(^2\).
- Max blocking: ~WCRT of work-group.
- Swap: HW+OpenCL configuration.

\(^2\) Tanasic et al. “Enabling preemptive multiprogramming on GPUs, 2014”
Context switch mechanisms and preemption models

Our claim: SM draining, modelled by limited-preemptive scheduling, provides a good trade-off point for GPUs between:

- Context switching cost, and
- WCRT benefits.
Measure context switch response time

“The Fermi pipeline is optimized to reduce the cost of an application context switch to below 25 microseconds.”

---

Measure context switch response time

“The Fermi pipeline is optimized to reduce the cost of an application 
context switch to below 25 microseconds.”

- Is 25µs an average or worst-case time?
- Is 25µs execution time or response time?
- What is the distribution?

---

Measure context switch response time - experiment

Characterise WCRT of hardware (non-preemptive) context switch.

Approach:

1. Modify (nouveau’s) context switching firmware to report WCRT.
   - **Excluding** time to finish current kernel execution.
   - Intrusive measurement, max. observed overhead 224ns.
Characterise WCRT of hardware (non-preemptive) context switch.

Approach:

1. Modify (nouveau’s) context switching firmware to report WCRT.
   - Excluding time to finish current kernel execution.
   - Intrusive measurement, max. observed overhead 224ns.

2. Write program to read from hardware:
   - Context size,
   - Reported context switch time.
Measure context switch response time - experiment

Characterise WCRT of hardware (non-preemptive) context switch.

Approach:

1. Modify (nouveau’s) context switching firmware to report WCRT.
   - Excluding time to finish current kernel execution.
   - Intrusive measurement, max. observed overhead 224ns.

2. Write program to read from hardware:
   - Context size,
   - Reported context switch time.

3. For several Kepler GPUs (2012-2014) gather 20M samples each.
   - 1600x1200 X.org/XFCE desktop,
   - 1024x768 OpenArena windowed timedemo.
Measure context switch response time - results

<table>
<thead>
<tr>
<th></th>
<th>SM</th>
<th>Cores</th>
<th>Max bw</th>
<th>State</th>
<th>Time (µs)</th>
<th>Avg. bw</th>
<th>Util.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeForce</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min</td>
<td>Avg</td>
<td>Max</td>
</tr>
<tr>
<td>GT 710</td>
<td>1</td>
<td>953</td>
<td>14.4</td>
<td>63.9</td>
<td>9.2</td>
<td>21.5</td>
<td>80.1</td>
</tr>
<tr>
<td>GT 640</td>
<td>2</td>
<td>901</td>
<td>28.5</td>
<td>68.2</td>
<td>13.6</td>
<td>26.5</td>
<td>43.7</td>
</tr>
<tr>
<td>GTX 650</td>
<td>2</td>
<td>1058</td>
<td>80.0</td>
<td>68.2</td>
<td>12.7</td>
<td>23.2</td>
<td>36.0</td>
</tr>
<tr>
<td>GTX 780</td>
<td>12</td>
<td>992</td>
<td>288.4</td>
<td>268.6</td>
<td>9.7</td>
<td>20.0</td>
<td>28.6</td>
</tr>
</tbody>
</table>

- What is the average context switch time? 20.0 – 26.5µs.
- What is the worst-case context switch time? > 28.6µs.
### Measure context switch response time - results

<table>
<thead>
<tr>
<th>NVIDIA GeForce</th>
<th>SM</th>
<th>Cores</th>
<th>Max bw</th>
<th>State</th>
<th>Time (µs)</th>
<th>Avg. bw</th>
<th>Util.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT 710</td>
<td>1</td>
<td>953</td>
<td>14.4</td>
<td>63.9</td>
<td>9.2</td>
<td>21.5</td>
<td>80.1</td>
</tr>
<tr>
<td>GT 640</td>
<td>2</td>
<td>901</td>
<td>28.5</td>
<td>68.2</td>
<td>13.6</td>
<td>26.5</td>
<td>43.7</td>
</tr>
<tr>
<td>GTX 650</td>
<td>2</td>
<td>1058</td>
<td>80.0</td>
<td>68.2</td>
<td>12.7</td>
<td>23.2</td>
<td>36.0</td>
</tr>
<tr>
<td>GTX 780</td>
<td>12</td>
<td>992</td>
<td>288.4</td>
<td>268.6</td>
<td>9.7</td>
<td>20.0</td>
<td>28.6</td>
</tr>
</tbody>
</table>

- What is the average context switch time? 20.0 – 26.5µs.
- What is the worst-case context switch time? > 28.6µs.
- Execution time or response time?
  - Ex. time: Average context switch time not strictly memory bound.
Measure context switch response time - results

<table>
<thead>
<tr>
<th>NVIDIA GeForce</th>
<th>SM</th>
<th>Cores</th>
<th>Max bw</th>
<th>State</th>
<th>Time (µs)</th>
<th>Avg. bw</th>
<th>Util.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT 710</td>
<td>1</td>
<td>953</td>
<td>14.4</td>
<td>63.9</td>
<td>9.2</td>
<td>21.5</td>
<td>80.1</td>
</tr>
<tr>
<td>GT 640</td>
<td>2</td>
<td>901</td>
<td>28.5</td>
<td>68.2</td>
<td>13.6</td>
<td>26.5</td>
<td>43.7</td>
</tr>
<tr>
<td>GTX 650</td>
<td>2</td>
<td>1058</td>
<td>80.0</td>
<td>68.2</td>
<td>12.7</td>
<td>23.2</td>
<td>36.0</td>
</tr>
<tr>
<td>GTX 780</td>
<td>12</td>
<td>992</td>
<td>288.4</td>
<td>268.6</td>
<td>9.7</td>
<td>20.0</td>
<td>28.6</td>
</tr>
</tbody>
</table>

- What is the average context switch time? $20.0 - 26.5\mu s$.
- What is the worst-case context switch time? $>28.6\mu s$.
- Execution time or response time?
  - Ex. time: Average context switch time not strictly memory bound.
  - Resp. time: Worst case overhead due to interference on DRAM bus from display scan-out.
Measure context switch response time - results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min</td>
<td>Avg</td>
<td>Max</td>
</tr>
<tr>
<td>GT 710</td>
<td>1</td>
<td>953</td>
<td>14.4</td>
<td>63.9</td>
<td>9.2</td>
<td>21.5</td>
<td>80.1</td>
</tr>
<tr>
<td>GT 640</td>
<td>2</td>
<td>901</td>
<td>28.5</td>
<td>68.2</td>
<td>13.6</td>
<td>26.5</td>
<td>43.7</td>
</tr>
<tr>
<td>GTX 650</td>
<td>2</td>
<td>1058</td>
<td>80.0</td>
<td>68.2</td>
<td>12.7</td>
<td>23.2</td>
<td>36.0</td>
</tr>
<tr>
<td>GTX 780</td>
<td>12</td>
<td>992</td>
<td>288.4</td>
<td>268.6</td>
<td>9.7</td>
<td>20.0</td>
<td>28.6</td>
</tr>
</tbody>
</table>

- What is the average context switch time? $20.0 - 26.5\mu s$.
- What is the worst-case context switch time? $> 28.6\mu s$.

- Execution time or response time?
  - Ex. time: Average context switch time not strictly memory bound.
  - Resp. time: Worst case overhead due to interference on DRAM bus from display scan-out.

- Distribution (GT 710): 0.3% of samples in $[23.6, \infty]$.  
  - see paper for plot.
Preemption models

Our claim: SM draining, modelled by limited-preemptive scheduling, provides a good trade-off point for GPUs between:

- Context switching cost, and
  - WCRT benefits.
Task scheduling on GPUs - experiment

Study WCRT implications of scheduling models under context switching constraints, through overhead-aware schedulability experiment.

Approach:
1. Determine feasible parameters/ranges for
   - Context switch overheads for different scheduling policies,
   - (Periodic) task sets.
2. Compare schedulability of random task sets.
## Task scheduling on GPUs - parameters

<table>
<thead>
<tr>
<th>Scheduling policy</th>
<th>Ctx</th>
<th>Reg</th>
<th>Local</th>
<th>Total</th>
<th>Time (µs)</th>
<th>Preempt /job&lt;sup&gt;4&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full preemptive (EDF)</td>
<td>68.2</td>
<td>512</td>
<td>96</td>
<td>676.2</td>
<td>263</td>
<td>×2</td>
</tr>
<tr>
<td>SM draining (lpEDF)</td>
<td>68.2</td>
<td>0</td>
<td>0</td>
<td>68.2</td>
<td>27</td>
<td>×2</td>
</tr>
<tr>
<td>Non-preemptive (npEDF)</td>
<td>68.2</td>
<td>0</td>
<td>0</td>
<td>68.2</td>
<td>27</td>
<td>×1</td>
</tr>
</tbody>
</table>

(Based on GeForce GT 640 (2×SM), resembling Tegra K1)

---

## Task scheduling on GPUs - parameters

<table>
<thead>
<tr>
<th>Scheduling policy</th>
<th>State (KiB)</th>
<th>Time (µs)</th>
<th>Preempt/job$^4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ctx</td>
<td>Reg</td>
<td>Local</td>
</tr>
<tr>
<td>Full preemptive (EDF)</td>
<td>68.2</td>
<td>512</td>
<td>96</td>
</tr>
<tr>
<td>SM draining (lpEDF)</td>
<td>68.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Non-preemptive (npEDF)</td>
<td>68.2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(Based on GeForce GT 640 (2×SM), resembling Tegra K1)

Linear correlation state size $\leftrightarrow$ context switch time

---

## Task scheduling on GPUs - parameters

<table>
<thead>
<tr>
<th>Scheduling policy</th>
<th>State (KiB)</th>
<th>Time (µs)</th>
<th>Preempt /job$^4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ctx Reg Local Total Avg Max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full preemptive (EDF)</td>
<td>68.2 512 96 676.2</td>
<td>263 434</td>
<td>×2</td>
</tr>
<tr>
<td>SM draining (lpEDF)</td>
<td>68.2 0 0 68.2</td>
<td>27 44</td>
<td>×2</td>
</tr>
<tr>
<td>Non-preemptive (npEDF)</td>
<td>68.2 0 0 68.2</td>
<td>27 44</td>
<td>×1</td>
</tr>
</tbody>
</table>

(Based on GeForce GT 640 (2×SM), resembling Tegra K1)

Linear correlation state size ↔ context switch time

Inflate task cost with n × context switch time

---

Preemptive GPU scheduling

Compare schedulability of random task sets:

Task set:
- Uniprocessor EDF scheduling policy.
- $U = \{0.2, 0.21, \ldots, 1.0\}$
- $100,000 \times 81 = 8.1$M random task sets (UUniFast).
- Task set: two tasks, $1,000 \mu s \leq P_i < 15,000 \mu s$.
- lpEDF: max blocking $q = \frac{c}{\text{random}(2,500)}$, 2-500 WGs per SM.
Preemptive GPU scheduling

For 0.25 \( U \geq 0.72 \), full-preempt is beneficial. Reducing preemptive context switch overhead results in higher schedulability.
Preemptive GPU scheduling

For $0.25 \leq U \leq 0.72$ full-preempt beneficial
Preemptive GPU scheduling

For $0.25 \leq U \leq 0.72$ full-preempt beneficial

Reduce preemptive ctxswitch overhead → higher schedulability.
Preemptive GPU scheduling

Limited-preemption far outperforms other models!
Limited-preemptive scheduling (SM draining) provides a good trade-off point for GPUs between context switching cost and WCRT benefits.

- Current GPUs: context switch 20 – 26.5μs on average.
- Overhead-aware schedulability experiment demonstrates advantage of SM draining model.
Limited-preemptive scheduling (SM draining) provides a good trade-off point for GPUs between context switching cost and WCRT benefits.

- Current GPUs: context switch 20 – 26.5μs on average.
- Overhead-aware schedulability experiment demonstrates advantage of SM draining model.

In the paper:
- Histogram of context switch times GeForce GT 710.
- Demonstration of interference context switch ↔ scan-out.
- Schedulability experiment with 3-task systems.
NVIDIA GPU architecture - streaming multiprocessor

**Streaming Multiprocessor (SM), simplified**
- Warp scheduler
- Warp scheduler
- Warp scheduler
- Warp scheduler

**Register file** (65536 * 32 bits = 256KiB)

- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Core
- Co
NVIDIA GPU architecture - FECS and GPCCS

GeForce GT 610

µc FECS

µc GPCCS

GPC

SM

GeForce GTX 780

µc FECS

µc GPCCS

GPC

SM SM SM

GPC

SM SM

GPC

SM SM

GPC

SM SM
NVIDIA GPU architecture - FECS and GPCCS

GeForce GT 610

µ
μ
κ
κ
FECS

GPC

µ
μ
κ
κ
GPCCS

SM

SM

SM

GeForce GTX 780

µ
μ
κ
κ
FECS

GPC

µ
μ
κ
κ
GPCCS

GPC

SM

SM

SM

SM

SM

SM

SM

Graphics Processing Cluster
NVIDIA GPU architecture - FECS and GPCCS

GeForce GT 610

- \( \mu c \) FECS
- GPC
- \( \mu c \) GPCCS

GeForce GTX 780

- GPC
- \( \mu c \) GPCCS
- SM
- SM
- SM

- GPC
- \( \mu c \) GPCCS
- SM
- SM

- GPC
- \( \mu c \) GPCCS
- SM
- SM

- GPC
- \( \mu c \) GPCCS
- SM
- SM

- GPC
- \( \mu c \) GPCCS
- SM
- SM

Graphics Processing Cluster

GPC Context Switch
NVIDIA GPU architecture - FECS and GPCCS

GeForce GT 610

- μc FECS
- SM
- μc GPCCS

GeForce GTX 780

- GPC
- μc GPCCS
- SM
- SM
- SM
- μc FECS

- GPC
- μc GPCCS
- SM
- SM
- SM

- GPC
- μc GPCCS
- SM
- SM

- GPC
- μc GPCCS
- SM
- SM

Graphics Processing Cluster

GPC Context Switch

Front-End Context Switch