BEST PRACTICES WHEN BENCHMARKING CUDA APPLICATIONS

Bill Fiser – Senior System Software Engineer
Sebastian Jodłowski – Senior System Software Engineer

AGENDA

Peak performance

vs.

Stable performance

AGENDA

Peak performance

vs.

Stable performance

AGENDA

System stability

- CPU Frequency Scaling
- NUMA
- GPU clocks

Measuring the right thing

- JIT cache
- CUDA events
- API contention

SYSTEM STABILITY

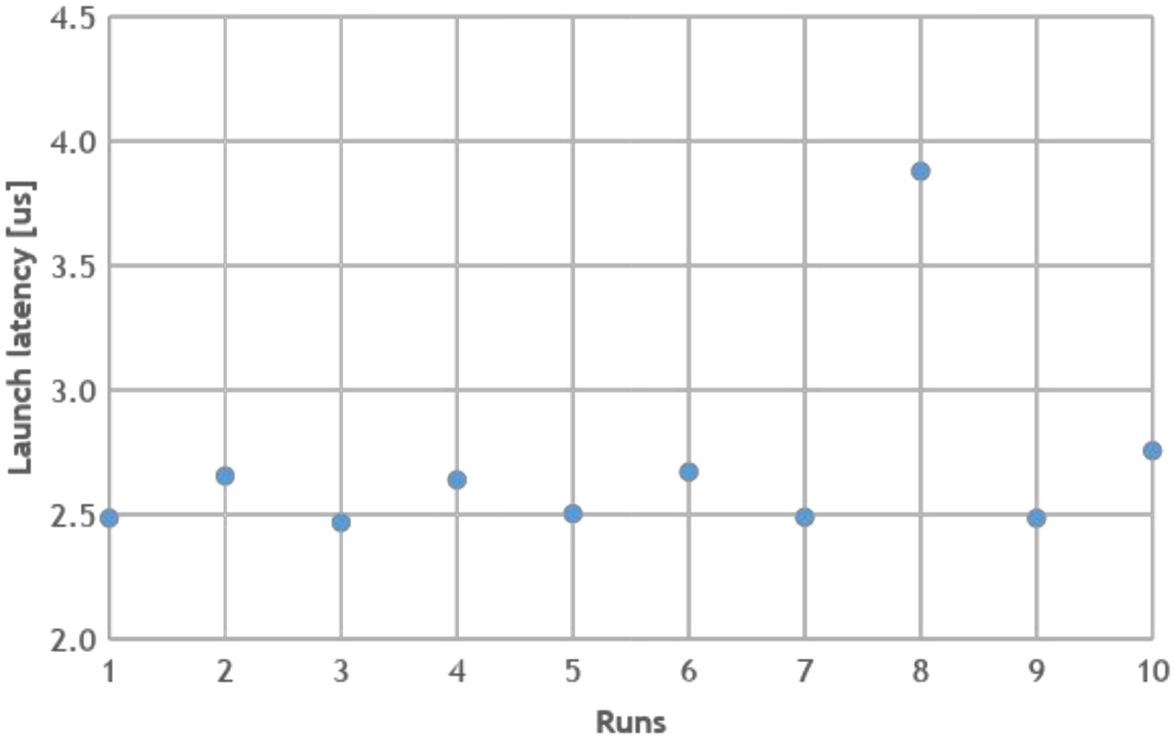
CPU FREQUENCY SCALING

Achieving Stable CPU Benchmarks: launch latency

```
#include <chrono>
#include <iostream>

using namespace std;
using namespace std::chrono;

__global__ void empty() {}

int main() {
    const int iters = 1000;

    cudaFree(0);
    empty<<<1,1>>>();
    cudaDeviceSynchronize();

    // Warmup phase
    for (int i = 0; i < 10; ++i) {
        empty<<<1,1>>>();
    }

    // Benchmark phase
    auto start = steady_clock::now();
    for (int i = 0; i < iters; ++i) {
        empty<<<1,1>>>();
    }
    auto end = steady_clock::now();

    auto usecs = duration_cast<duration<float,
        microseconds::period> >(end - start);
    cout << usecs.count() / iters << endl;
}
```

CPU FREQUENCY SCALING

Achieving Stable CPU Benchmarks: launch latency



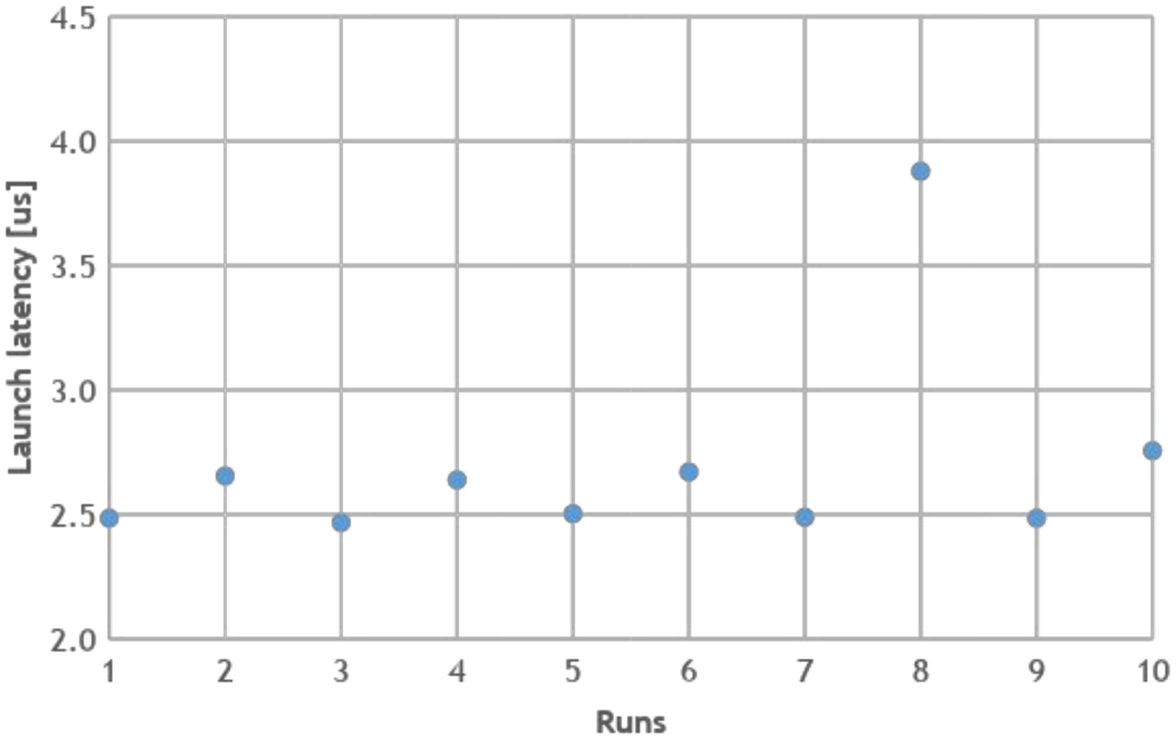
DGX-1V, Intel Xeon E5-2698 @ 2.20GHz

Average Launch Latency – 2.70 us

Relative Standard Deviation – 16%

CPU FREQUENCY SCALING

Achieving Stable CPU Benchmarks: launch latency



DGX-1V, Intel Xeon E5-2698 @ 2.20GHz

CPU clocks can fluctuate significantly

- This can be a result of CPU idling
- This can be a result of thermal or power throttling
- Can potentially cause unstable benchmark results

Average Launch Latency – 2.70 us

Relative Standard Deviation – 16%

CPU FREQUENCY SCALING

Monitoring Clocks and Policies

Using cpupower to monitor clocks while the test is running can reveal what is happening

```
user@dgx-1v:~$ cpupower monitor -m Mperf

               |Mperf
PKG |CORE|CPU | C0    | Cx    | Freq
   0|   0|   0| 99.13|  0.87|  3575
   0|   0|  40|  0.07| 99.93|  3360
   0|   1|   1|  9.64| 90.36|  3568
   0|   1|  41| 41.55| 58.45|  3576
   0|   2|   2|  0.05| 99.95|  2778
   0|   2|  42|  0.14| 99.86|  3249
   0|   3|   3|  0.06| 99.94|  2789
   0|   3|  43|  0.07| 99.93|  2835
   0|   4|   4|  0.07| 99.93|  2867
   0|   4|  44|  0.06| 99.94|  2912
   0|   8|   5|  0.05| 99.95|  2793
   0|   8|  45|  0.07| 99.93|  2905
```

```
user@dgx-1v:~$ cpupower frequency-info
analyzing CPU 0:
  driver: intel_pstate
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency:  Cannot determine or is not supported.
  hardware limits: 1.20 GHz - 3.60 GHz
  available cpufreq governors: performance powersave
  current policy: frequency should be within 1.20 GHz and 3.60 GHz.
                  The governor "powersave" may decide which speed to use
                  within this range.
  current CPU frequency: Unable to call hardware
  current CPU frequency: 1.31 GHz (asserted by call to kernel)
  boost state support:
    Supported: yes
    Active: yes
```

CPU FREQUENCY SCALING

Monitoring Clocks and Policies

CPU frequency scaling enables the operating system to scale the CPU frequency up or down in order to increase performance or save power

Scaling Governor set to “powersave” can result in CPU being underclocked longer than expected

Turbo Boost set to enabled can result in CPU being overclocked and eventually throttle

```
user@dgx-1v:~$ cpupower frequency-info
analyzing CPU 0:
  driver: intel_pstate
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency:  Cannot determine or is not supported.
  hardware limits: 1.20 GHz - 3.60 GHz
  available cpufreq governors: performance powersave
  current policy: frequency should be within 1.20 GHz and 3.60 GHz.
                  The governor "powersave" may decide which speed to use
                  within this range.
  current CPU frequency: Unable to call hardware
  current CPU frequency: 1.31 GHz (asserted by call to kernel)
  boost state support:
    Supported: yes
    Active: yes
```

CPU FREQUENCY SCALING

Achieving Stable CPU Benchmarks

With intel_pstate driver user cannot directly control CPU clocks

Use “performance” scaling governor and disable Turbo Boost for more **stable** benchmarking

```
user@dgx-1v:~$ # Set the Frequency Scaling Governor to Performance
user@dgx-1v:~$ sudo cpupower frequency-set -g performance
Setting cpu: 0
...
Setting cpu: 79
user@dgx-1v:~$ # Disable Turbo Boost
user@dgx-1v:~$ echo "1" | sudo tee
/sys/devices/system/cpu/intel_pstate/no_turbo
1
```

CPU FREQUENCY SCALING

Achieving Stable CPU Benchmarks

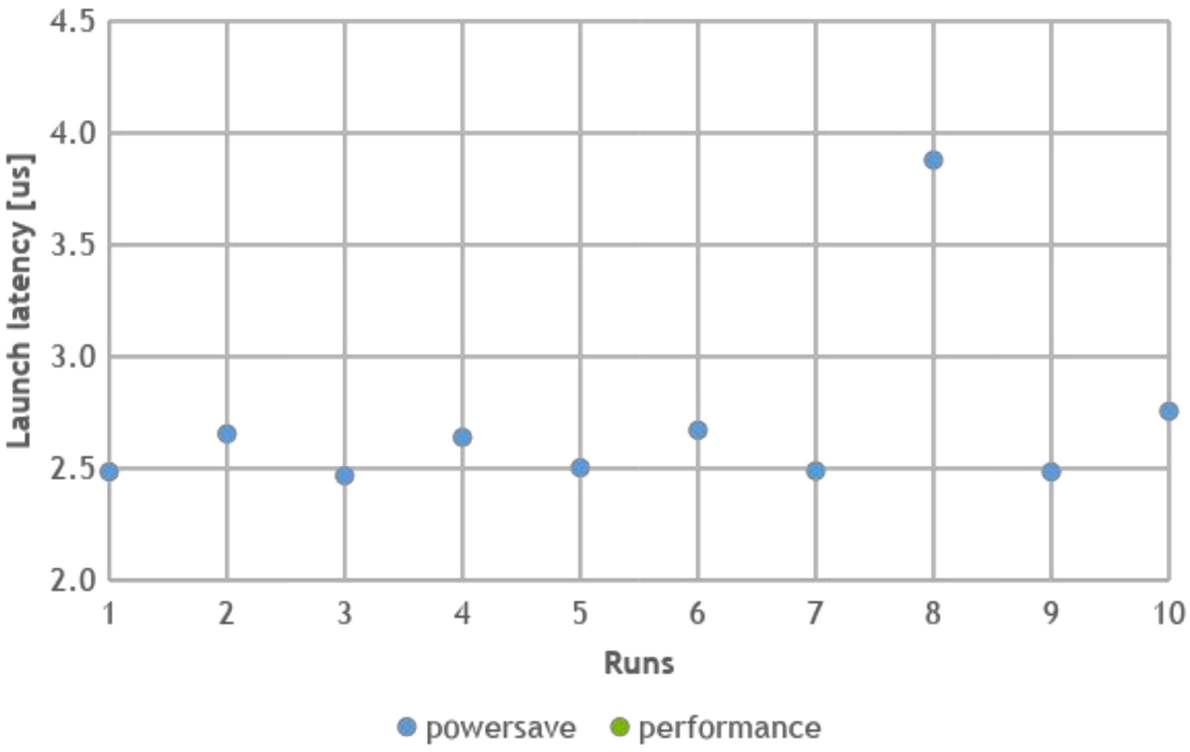
This helps keeping CPU clocks in more stable state

```
user@dgx-1v:~$ cpupower monitor –m Mperf

              |Mperf
PKG |CORE|CPU | C0    | Cx    | Freq
   0|   0|   0| 93.43|  6.57|  2192
   0|   0|  40|  0.45| 99.55|  2191
   0|   1|   1|  0.75| 99.25|  2185
   0|   1|  41|  0.60| 99.40|  2193
   0|   2|   2|  2.71| 97.29|  2192
   0|   2|  42|  0.56| 99.44|  2193
   0|   3|   3|  0.52| 99.48|  2193
   0|   3|  43|  0.53| 99.47|  2193
   0|   4|   4|  0.46| 99.54|  2193
   0|   4|  44|  0.56| 99.44|  2186
   0|   8|   5|  0.48| 99.52|  2193
   0|   8|  45|  0.54| 99.46|  2193
```

```
user@dgx-1v:~$ cpupower frequency-info
analyzing CPU 0:
  driver: intel_pstate
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency:  Cannot determine or is not supported.
  hardware limits: 1.20 GHz - 3.60 GHz
  available cpufreq governors: performance powersave
  current policy: frequency should be within 1.20 GHz and 2.20 GHz.
                  The governor "performance" may decide which speed to use
                  within this range.
  current CPU frequency: Unable to call hardware
  current CPU frequency: 2.19 GHz (asserted by call to kernel)
  boost state support:
    Supported: yes
    Active: yes
```

CPU FREQUENCY SCALING

Achieving Stable CPU Benchmarks: launch latency



DGX-1V, Intel Xeon E5-2698 @ 2.20GHz

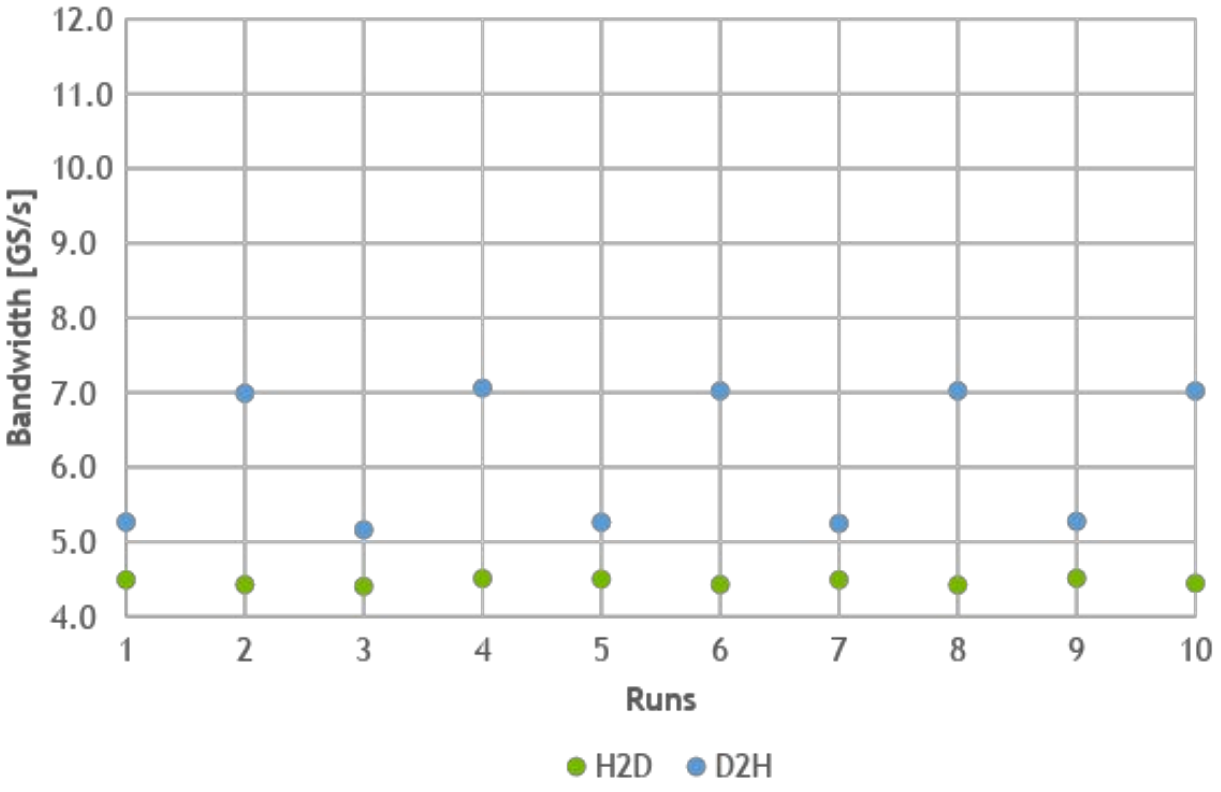
Better stability with
“performance” scaling governor

Average Launch Latency – 2.61 us

Relative Standard Deviation – 3%

NUMA

Achieving Stable Memory Benchmarks: pageable copies



DGX-1V, Intel Xeon E5-2698 @ 2.20GHz

<u>Host-to-device pageable memcopy:</u>

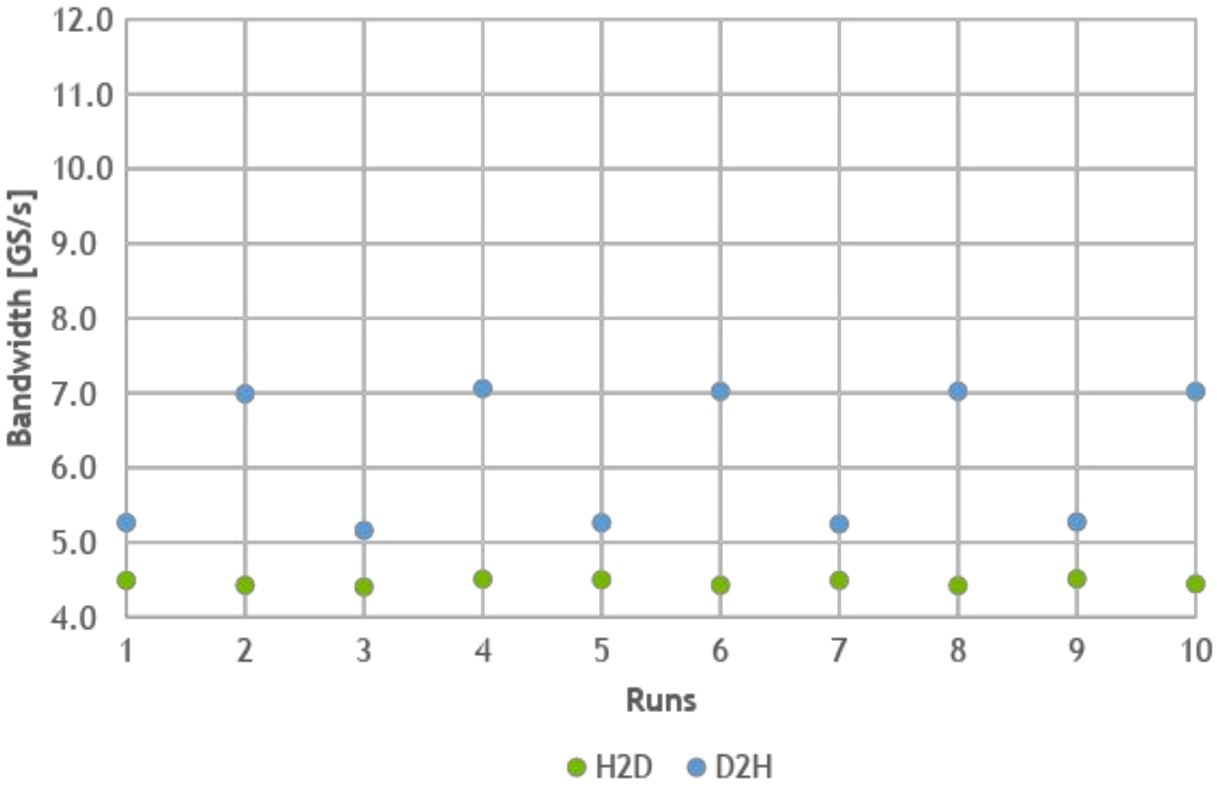
Average Bandwidth – 4.5 GB/s
Relative Standard Deviation – 1%

<u>Device-to-host pageable memcopy:</u>

Average Bandwidth – 6.1 GB/s
Relative Standard Deviation – 15%

NUMA

Achieving Stable Memory Benchmarks: pageable copies



DGX-1V, Intel Xeon E5-2698 @ 2.20GHz

Low or unstable bandwidth might be caused by CPU migrations or accesses to non-local memory.

Host-to-device pageable memcopy:

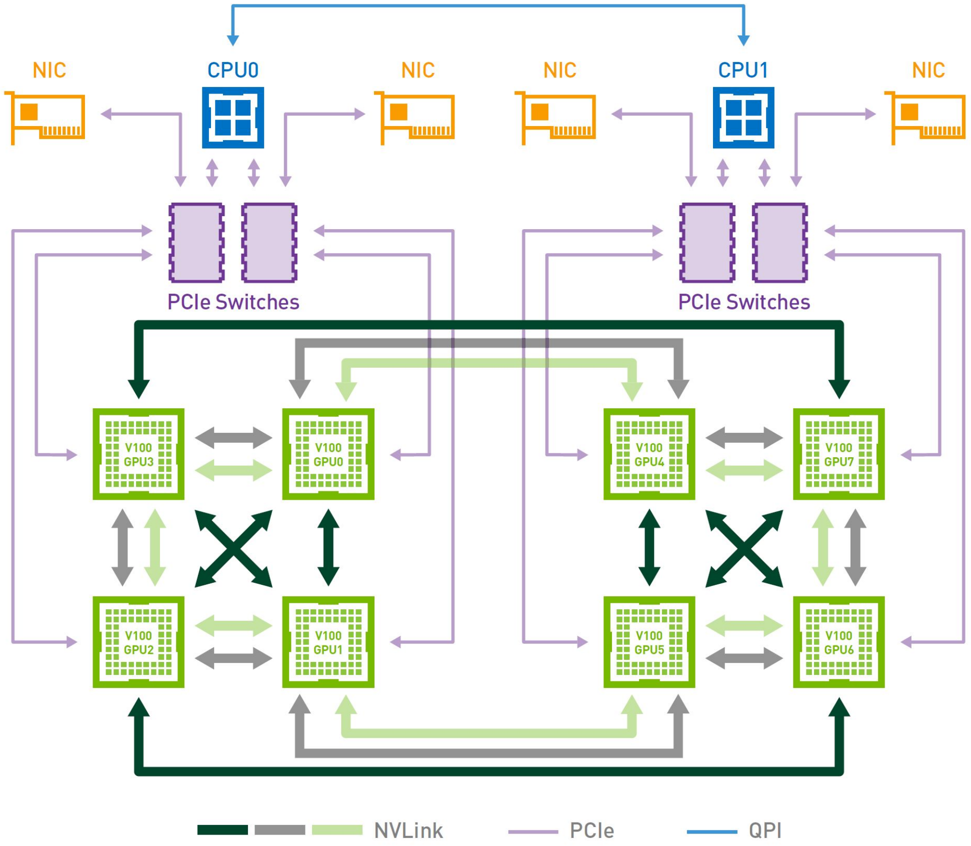
Average Bandwidth – 4.5 GB/s
Relative Standard Deviation – 1%

Device-to-host pageable memcopy:

Average Bandwidth – 6.1 GB/s
Relative Standard Deviation – 15%

NUMA

DGX-1V Topology



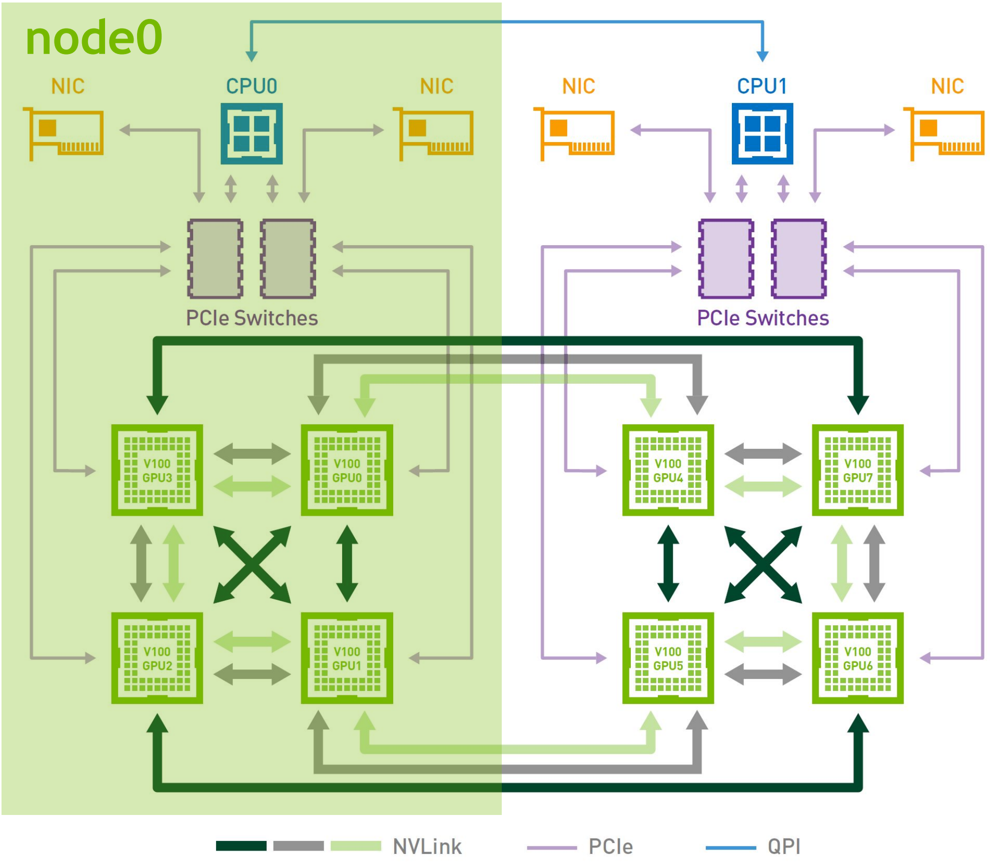
Non-Uniform Memory Access (NUMA) allows system memory to be divided into zones (nodes)

NUMA nodes are allocated to particular CPUs or sockets

Memory bandwidth and latencies between NUMA nodes might not be the same

NUMA

DGX-1V Topology



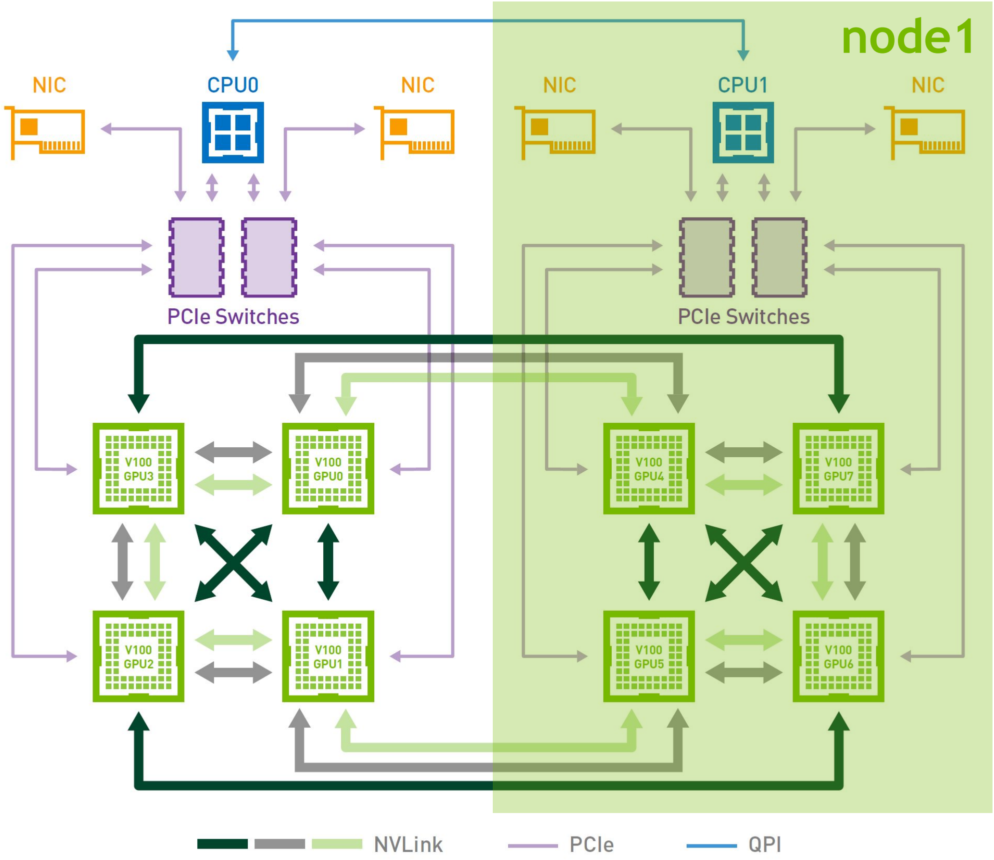
Non-Uniform Memory Access (NUMA) allows system memory to be divided into zones (nodes)

NUMA nodes are allocated to particular CPUs or sockets

Memory bandwidth and latencies between NUMA nodes might not be the same

NUMA

DGX-1V Topology



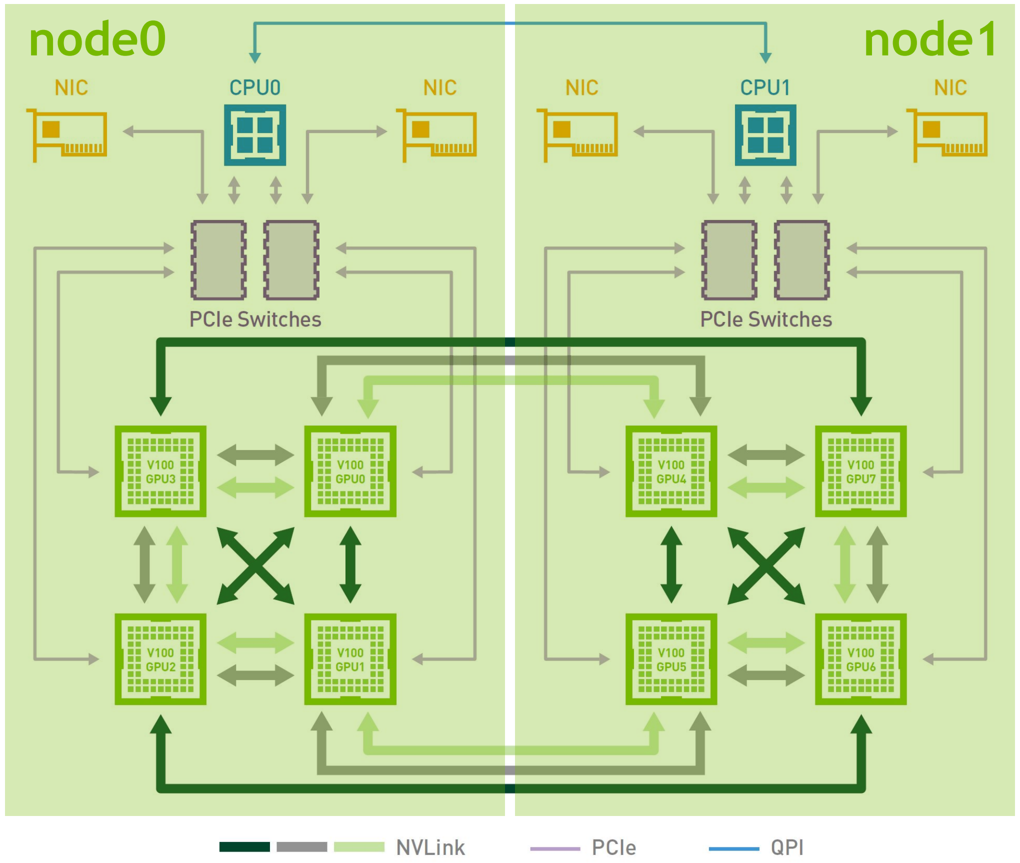
Non-Uniform Memory Access (NUMA) allows system memory to be divided into zones (nodes)

NUMA nodes are allocated to particular CPUs or sockets

Memory bandwidth and latencies between NUMA nodes might not be the same

NUMA

DGX-1V Topology



Non-Uniform Memory Access (NUMA) allows system memory to be divided into zones (nodes)

NUMA nodes are allocated to particular CPUs or sockets

Memory bandwidth and latencies between NUMA nodes might not be the same

NUMA

Querying NUMA configuration

Use numactl to check NUMA nodes configuration

```
user@dgx-1v:~$ numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59
node 0 size: 257844 MB
node 0 free: 255674 MB
node 1 cpus: 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 60 61 62 63 64 65 66 67 68 69 70 71 72 73
74 75 76 77 78 79
node 1 size: 258039 MB
node 1 free: 256220 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

NUMA

Querying NUMA configuration

Use numactl to check NUMA nodes configuration

```
user@dgx-1v:~$ numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59
node 0 size: 257844 MB
node 0 free: 255674 MB
node 1 cpus: 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 60 61 62 63 64 65 66 67 68 69 70 71 72 73
74 75 76 77 78 79
node 1 size: 258039 MB
node 1 free: 256220 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

NUMA

Querying NUMA configuration

Use numactl to check NUMA nodes configuration

```
user@dgx-1v:~$ numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59
node 0 size: 257844 MB
node 0 free: 255674 MB
node 1 cpus: 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 60 61 62 63 64 65 66 67 68 69 70 71 72 73
74 75 76 77 78 79
node 1 size: 258039 MB
node 1 free: 256220 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

NUMA

Querying NUMA configuration

Use numactl to check NUMA nodes configuration

```
user@dgx-1v:~$ numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59
node 0 size: 257844 MB
node 0 free: 255674 MB
node 1 cpus: 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 60 61 62 63 64 65 66 67 68 69 70 71 72 73
74 75 76 77 78 79
node 1 size: 258039 MB
node 1 free: 256220 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

NUMA

Querying System Topology

Use nvidia-smi to check which CPU is the closest to the given GPU

```
user@dgx-1v:~$ nvidia-smi topo -mp
        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7    mlx5_1  mlx5_2  mlx5_3  mlx5_0  CPU Affinity
GPU0     X      PIX     PHB     PHB     SYS     SYS     SYS     SYS     PHB     SYS     SYS     PIX     0-19,40-59
GPU1    PIX      X      PHB     PHB     SYS     SYS     SYS     SYS     PHB     SYS     SYS     PIX     0-19,40-59
GPU2    PHB     PHB      X      PIX     SYS     SYS     SYS     SYS     PIX     SYS     SYS     PHB     0-19,40-59
GPU3    PHB     PHB     PIX      X      SYS     SYS     SYS     SYS     PIX     SYS     SYS     PHB     0-19,40-59
GPU4    SYS     SYS     SYS     SYS      X      PIX     PHB     PHB     SYS     PIX     PHB     SYS     20-39,60-79
GPU5    SYS     SYS     SYS     SYS     PIX      X      PHB     PHB     SYS     PIX     PHB     SYS     20-39,60-79
GPU6    SYS     SYS     SYS     SYS     PHB     PHB      X      PIX     SYS     PHB     PIX     SYS     20-39,60-79
GPU7    SYS     SYS     SYS     SYS     PHB     PHB     PIX      X      SYS     PHB     PIX     SYS     20-39,60-79
mlx5_1  PHB     PHB     PIX     PIX     SYS     SYS     SYS     SYS      X      SYS     SYS     PHB
mlx5_2  SYS     SYS     SYS     SYS     PIX     PIX     PHB     PHB     SYS      X      PHB     SYS
mlx5_3  SYS     SYS     SYS     SYS     PHB     PHB     PIX     PIX     SYS     PHB      X      SYS
mlx5_0  PIX     PIX     PHB     PHB     SYS     SYS     SYS     SYS     PHB     SYS     SYS      X
```

NUMA

Querying System Topology

Use nvidia-smi to check which CPU is the closest to the given GPU

```
user@dgx-1v:~$ nvidia-smi topo -mp
        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7    mlx5_1  mlx5_2  mlx5_3  mlx5_0  CPU Affinity
GPU0     X      PIX     PHB     PHB     SYS     SYS     SYS     SYS     PHB     SYS     SYS     PIX     0-19,40-59
GPU1    PIX      X      PHB     PHB     SYS     SYS     SYS     SYS     PHB     SYS     SYS     PIX     0-19,40-59
GPU2    PHB     PHB      X      PIX     SYS     SYS     SYS     SYS     PIX     SYS     SYS     PHB     0-19,40-59
GPU3    PHB     PHB     PIX      X      SYS     SYS     SYS     SYS     PIX     SYS     SYS     PHB     0-19,40-59
GPU4    SYS     SYS     SYS     SYS      X      PIX     PHB     PHB     SYS     PIX     PHB     SYS     20-39,60-79
GPU5    SYS     SYS     SYS     SYS     PIX      X      PHB     PHB     SYS     PIX     PHB     SYS     20-39,60-79
GPU6    SYS     SYS     SYS     SYS     PHB     PHB      X      PIX     SYS     PHB     PIX     SYS     20-39,60-79
GPU7    SYS     SYS     SYS     SYS     PHB     PHB     PIX      X      SYS     PHB     PIX     SYS     20-39,60-79
mlx5_1  PHB     PHB     PIX     PIX     SYS     SYS     SYS     SYS      X      SYS     SYS     PHB
mlx5_2  SYS     SYS     SYS     SYS     PIX     PIX     PHB     PHB     SYS      X      PHB     SYS
mlx5_3  SYS     SYS     SYS     SYS     PHB     PHB     PIX     PIX     SYS     PHB      X      SYS
mlx5_0  PIX     PIX     PHB     PHB     SYS     SYS     SYS     SYS     PHB     SYS     SYS      X
```

NUMA

Querying System Topology

Use nvidia-smi to check which peer-GPUs belong to a different NUMA node

```
user@dgx-1v:~$ nvidia-smi topo -mp
        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7    mlx5_1  mlx5_2  mlx5_3  mlx5_0  CPU Affinity
GPU0     X      PIX     PHB     PHB     SYS     SYS     SYS     SYS     PHB     SYS     SYS     PIX     0-19,40-59
GPU1    PIX      X      PHB     PHB     SYS     SYS     SYS     SYS     PHB     SYS     SYS     PIX     0-19,40-59
GPU2    PHB     PHB      X      PIX     SYS     SYS     SYS     SYS     PIX     SYS     SYS     PHB     0-19,40-59
GPU3    PHB     PHB     PIX      X      SYS     SYS     SYS     SYS     PIX     SYS     SYS     PHB     0-19,40-59
GPU4    SYS     SYS     SYS     SYS      X      PIX     PHB     PHB     SYS     PIX     PHB     SYS     20-39,60-79
GPU5    SYS     SYS     SYS     SYS     PIX      X      PHB     PHB     SYS     PIX     PHB     SYS     20-39,60-79
GPU6    SYS     SYS     SYS     SYS     PHB     PHB      X      PIX     SYS     PHB     PIX     SYS     20-39,60-79
GPU7    SYS     SYS     SYS     SYS     PHB     PHB     PIX      X      SYS     PHB     PIX     SYS     20-39,60-79
mlx5_1  PHB     PHB     PIX     PIX     SYS     SYS     SYS     SYS      X      SYS     SYS     PHB
mlx5_2  SYS     SYS     SYS     SYS     PIX     PIX     PHB     PHB     SYS      X      PHB     SYS
mlx5_3  SYS     SYS     SYS     SYS     PHB     PHB     PIX     PIX     SYS     PHB      X      SYS
mlx5_0  PIX     PIX     PHB     PHB     SYS     SYS     SYS     SYS     PHB     SYS     SYS      X
```

NUMA

Querying System Topology

Use nvidia-smi to check which peer-GPUs belong to a different NUMA node

```
user@dgx-1v:~$ nvidia-smi topo -mp
        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7    mlx5_1  mlx5_2  mlx5_3  mlx5_0  CPU Affinity
GPU0     X      PIX     PHB     PHB     SYS     SYS     SYS     SYS     PHB     SYS     SYS     PIX     0-19,40-59
GPU1    PIX      X      PHB     PHB     SYS     SYS     SYS     SYS     PHB     SYS     SYS     PIX     0-19,40-59
GPU2    PHB     PHB      X      PIX     SYS     SYS     SYS     SYS     PIX     SYS     SYS     PHB     0-19,40-59
GPU3    PHB     PHB     PIX      X      SYS     SYS     SYS     SYS     PIX     SYS     SYS     PHB     0-19,40-59
GPU4    SYS     SYS     SYS     SYS      X      PIX     PHB     PHB     SYS     PIX     PHB     SYS     20-39,60-79
GPU5    SYS     SYS     SYS     SYS     PIX      X      PHB     PHB     SYS     PIX     PHB     SYS     20-39,60-79
GPU6    SYS     SYS     SYS     SYS     PHB     PHB      X      PIX     SYS     PHB     PIX     SYS     20-39,60-79
GPU7    SYS     SYS     SYS     SYS     PHB     PHB     PIX      X      SYS     PHB     PIX     SYS     20-39,60-79
mlx5_1  PHB     PHB     PIX     PIX     SYS     SYS     SYS     SYS      X      SYS     SYS     PHB
mlx5_2  SYS     SYS     SYS     SYS     PIX     PIX     PHB     PHB     SYS      X      PHB     SYS
mlx5_3  SYS     SYS     SYS     SYS     PHB     PHB     PIX     PIX     SYS     PHB      X      SYS
mlx5_0  PIX     PIX     PHB     PHB     SYS     SYS     SYS     SYS     PHB     SYS     SYS      X
```

NUMA

Achieving Stable Benchmarks

Use closest NUMA node for best **stability** (...and highest performance)

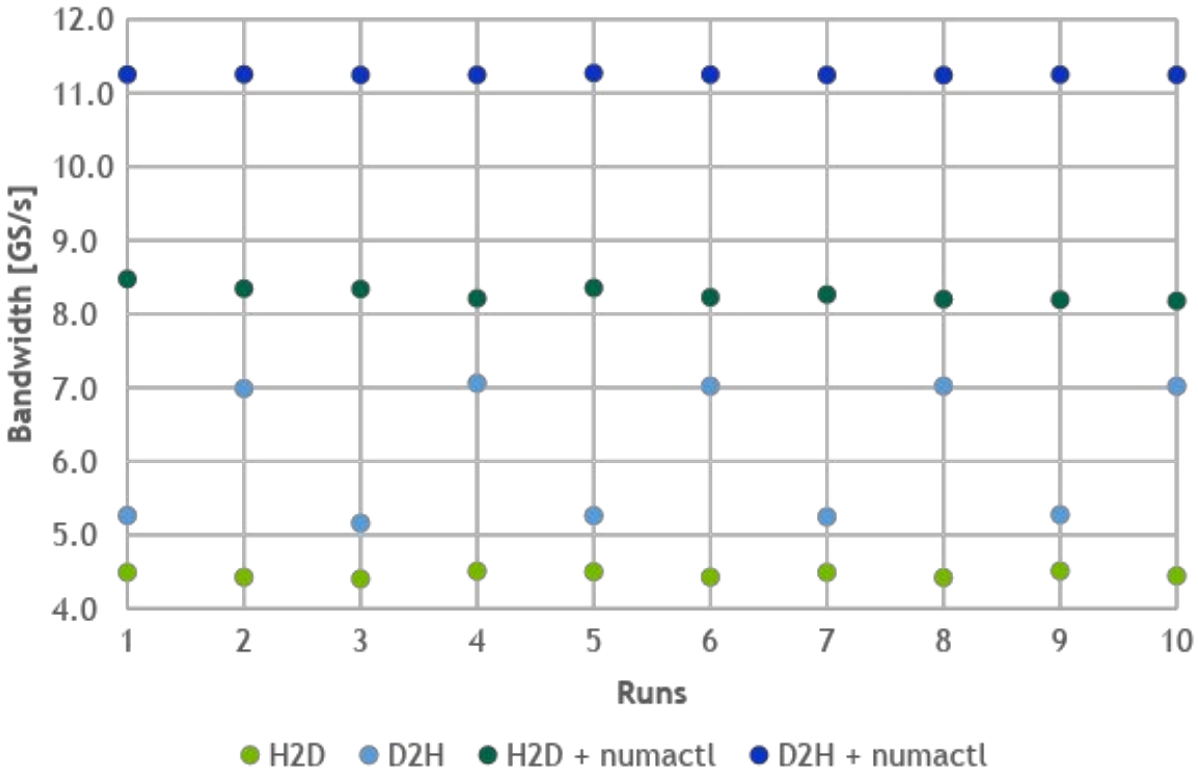
With numactl, you can set both:

- which NUMA node the application is executed on
- which NUMA node the application allocates memory from

```
user@dgx-1v:~$ numactl --cpunodebind=0 --membind=0 ./bandwidthTest --device=0
```

NUMA

Achieving Stable Memory Benchmarks: pageable copies



DGX-1V, Intel Xeon E5-2698 @ 2.20GHz

Better stability (...and performance) with correct NUMA setting

Host-to-device pageable memcopy:

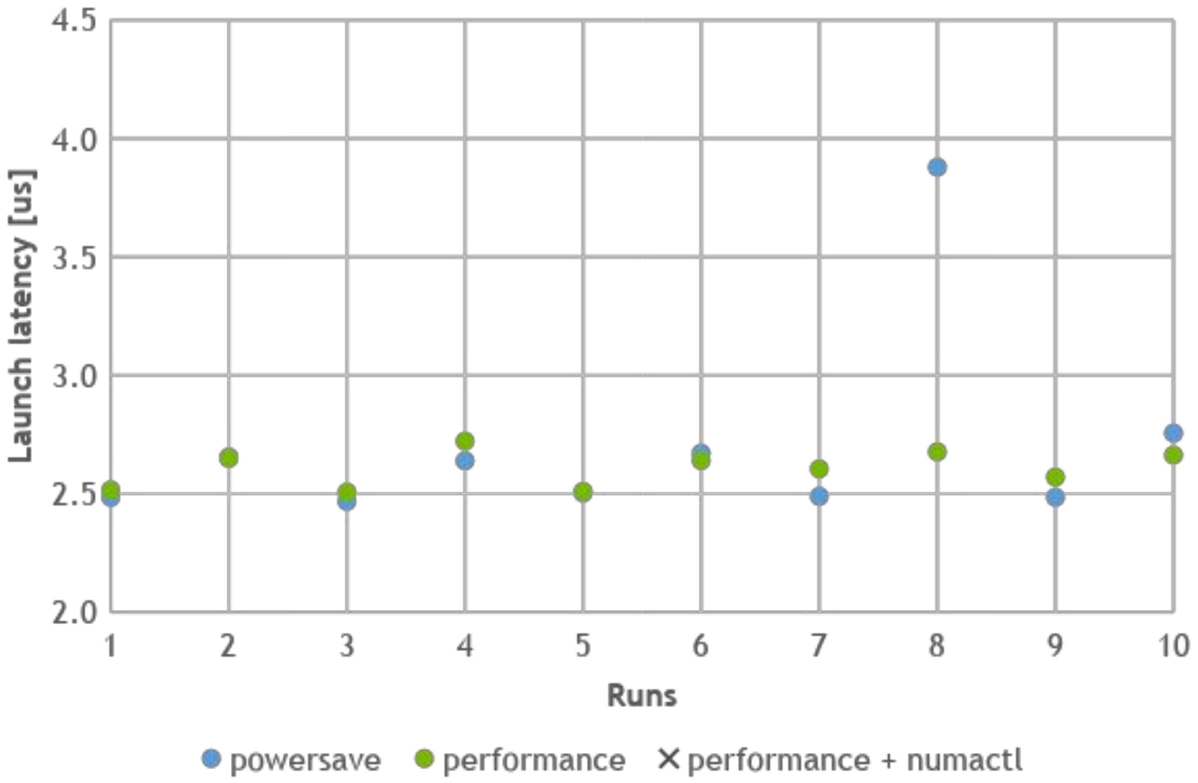
Average Bandwidth – 8.3 GB/s
Relative Standard Deviation – 1%

Device-to-host pageable memcopy:

Average Bandwidth – 11.3 GB/s
Relative Standard Deviation – 0%

NUMA

Achieving Stable CPU Benchmarks: launch latency



DGX-1V, Intel Xeon E5-2698 @ 2.20GHz

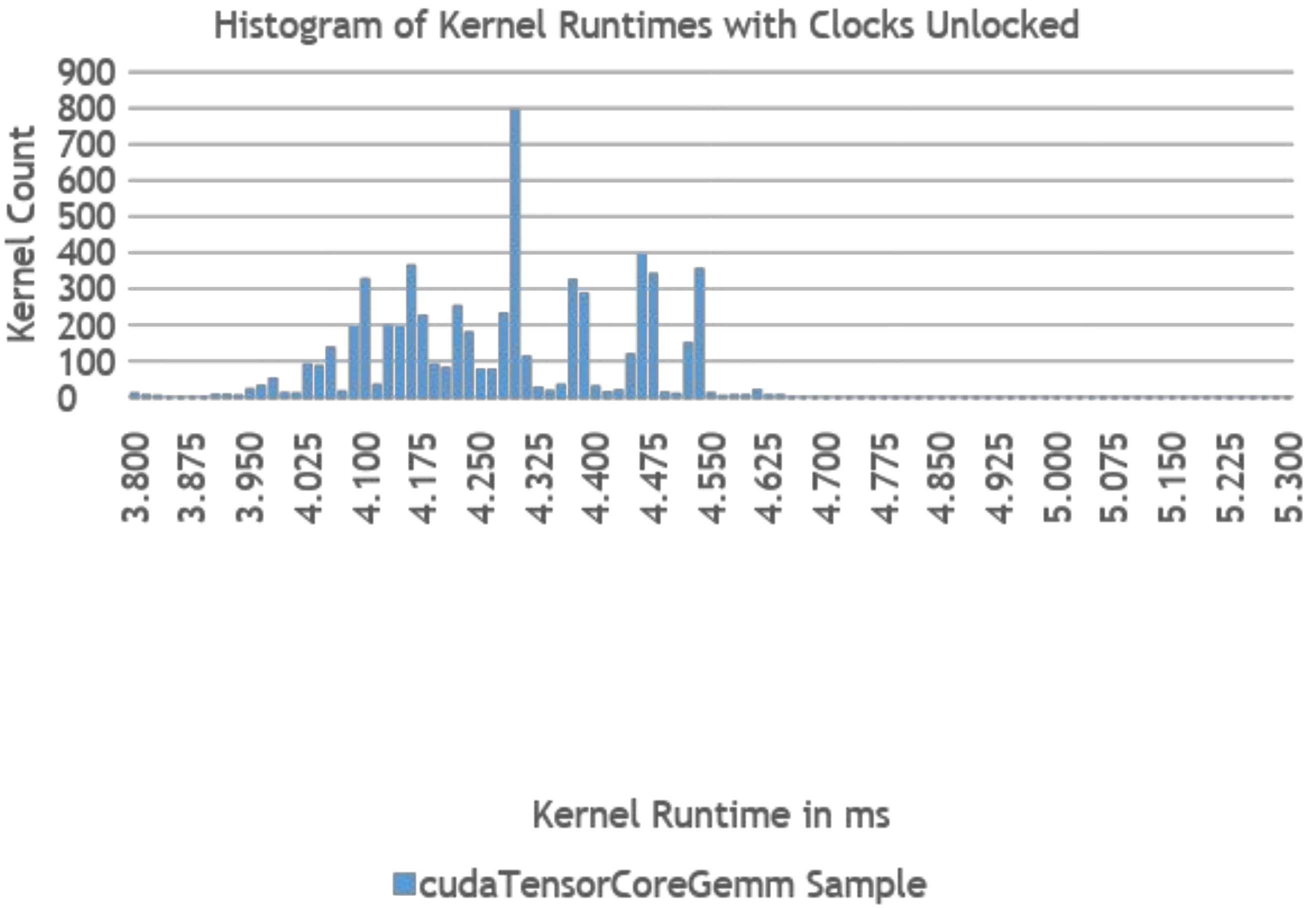
Better **stability** (...and performance) with
“performance” scaling governor
and
correct NUMA settings

Average Launch Latency – 2.47 us

Relative Standard Deviation – 1%

GPU CLOCK SETTINGS

Achieving Stable GPU Benchmarks



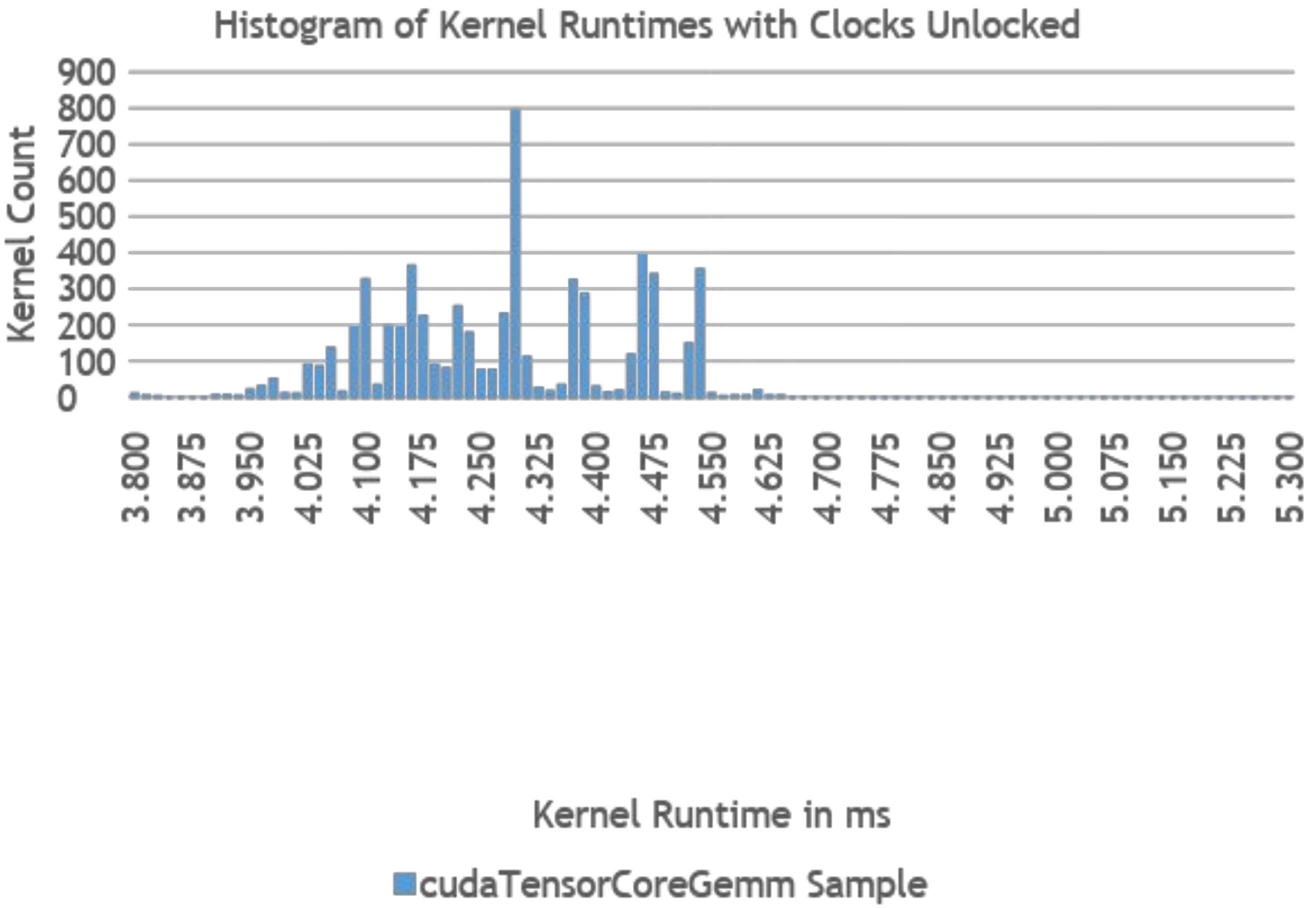
compute_gemm() kernel runtimes - RTX 4000

Mean Kernel Runtime – 4.27 ms

Relative Standard Deviation – 3.67%

GPU CLOCK SETTINGS

Achieving Stable GPU Benchmarks



Unrestricted, the GPU's clock settings can fluctuate significantly

- This can be a result of thermal or power throttling
- Can potentially cause unstable benchmark results

Mean Kernel Runtime – 4.27 ms

Relative Standard Deviation – 3.67%

GPU CLOCK SETTINGS

Monitoring Clocks and Throttling

Using nvidia-smi to monitor clocks while the test is running can reveal what is happening

- 'nvidia-smi -q -d PERFORMANCE' will show current Performance State and throttling
- 'nvidia-smi dmon' will scroll the current clock of the GPU

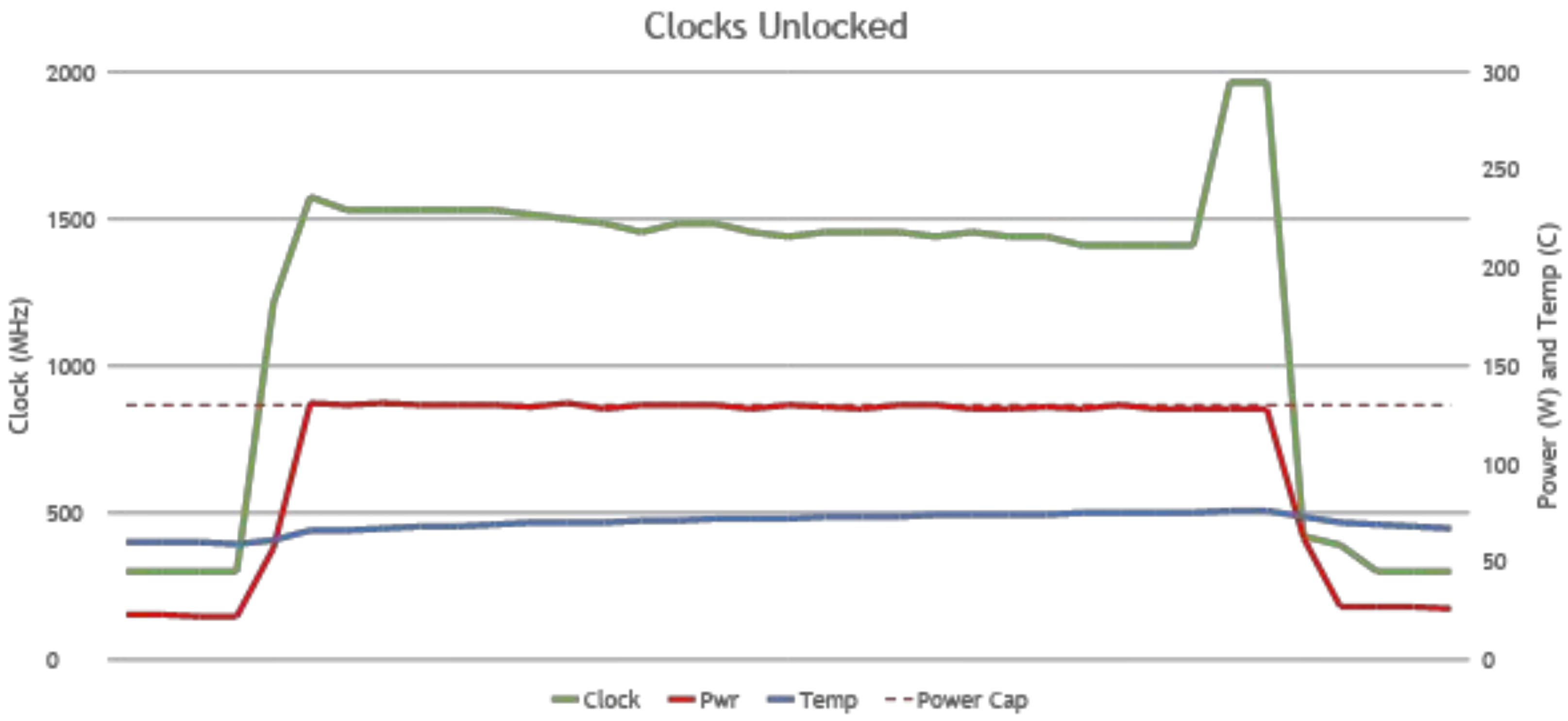
```
==============NVSMI LOG==============

Timestamp                           : Fri Feb 22 11:24:42 2019
Driver Version                      : 418.39
CUDA Version                        : 10.1

Attached GPUs                       : 1
GPU 00000000:01:00.0
    Performance State               : P0
    Clocks Throttle Reasons
        Idle                        : Not Active
        Applications Clocks Setting : Not Active
        SW Power Cap                : Active
        HW Slowdown                 : Not Active
            HW Thermal Slowdown     : Not Active
            HW Power Brake Slowdown : Not Active
        Sync Boost                  : Not Active
        SW Thermal Slowdown         : Not Active
        Display Clock Setting       : Not Active
```

# gpu	pwr	gtemp	mtemp	sm	mem	enc	dec	mclk	pclk
# Idx	W	C	C	%	%	%	%	MHz	MHz
0	22	59	-	0	0	0	0	405	300
0	57	61	-	1	0	0	0	6500	1215
0	131	66	-	22	12	0	0	6500	1575
0	130	68	-	100	66	0	0	6500	1530
0	130	69	-	100	66	0	0	6500	1530
0	129	70	-	100	65	0	0	6500	1515
0	131	70	-	100	65	0	0	6500	1500
0	128	70	-	100	65	0	0	6500	1485
0	130	71	-	100	64	0	0	6500	1455
0	130	72	-	100	64	0	0	6500	1485
0	128	72	-	100	64	0	0	6500	1455
0	130	72	-	100	63	0	0	6500	1440
0	130	73	-	100	62	0	0	6500	1455
0	130	74	-	100	62	0	0	6500	1440
0	128	74	-	100	62	0	0	6500	1455
0	129	74	-	100	62	0	0	6500	1440
0	130	75	-	100	62	0	0	6500	1410
0	128	75	-	100	62	0	0	6500	1410
0	128	76	-	100	61	0	0	6500	1965
0	128	76	-	60	38	0	0	6500	1965
0	62	73	-	0	0	0	0	6500	420

GPU CLOCK SETTINGS

Monitoring Clocks and Throttling

Using nvidia-smi to monitor clocks while the test is running can reveal what is happening



GPU CLOCK SETTINGS

Achieving Stable GPU Benchmarks

To achieve **stable** results, best practice is to lock the GPU's clock to default

- Clocks higher than default can be chosen, but monitor throttling with nvidia-smi
- 'nvidia-smi -q -d SUPPORTED_CLOCKS' lists available clock settings
- 'nvidia-smi -q -d CLOCK' shows current GPU clocks

```
==============NVSMI LOG==============

Timestamp                            : Fri Feb 22 11:27:21 2019
Driver Version                       : 418.39
CUDA Version                         : 10.1

Attached GPUs                        : 1
GPU 00000000:01:00.0
    Clocks
        Graphics                     : 300 MHz
        SM                           : 300 MHz
        Memory                       : 405 MHz
        Video                        : 540 MHz
    Applications Clocks
        Graphics                     : 1215 MHz
        Memory                       : 6501 MHz
    Default Applications Clocks
        Graphics                     : 1215 MHz
        Memory                       : 6501 MHz
    Max Clocks
        Graphics                     : 2100 MHz
        SM                           : 2100 MHz
        Memory                       : 6501 MHz
        Video                        : 1950 MHz
…
```

GPU CLOCK SETTINGS

Achieving Stable GPU Benchmarks

Use the values from “Default Application Clocks” for more stable benchmarking

- ‘nvidia-smi –ac <Default Memory Clock>,<Default Graphics Clock> to lock the clocks while an application is running on the GPU

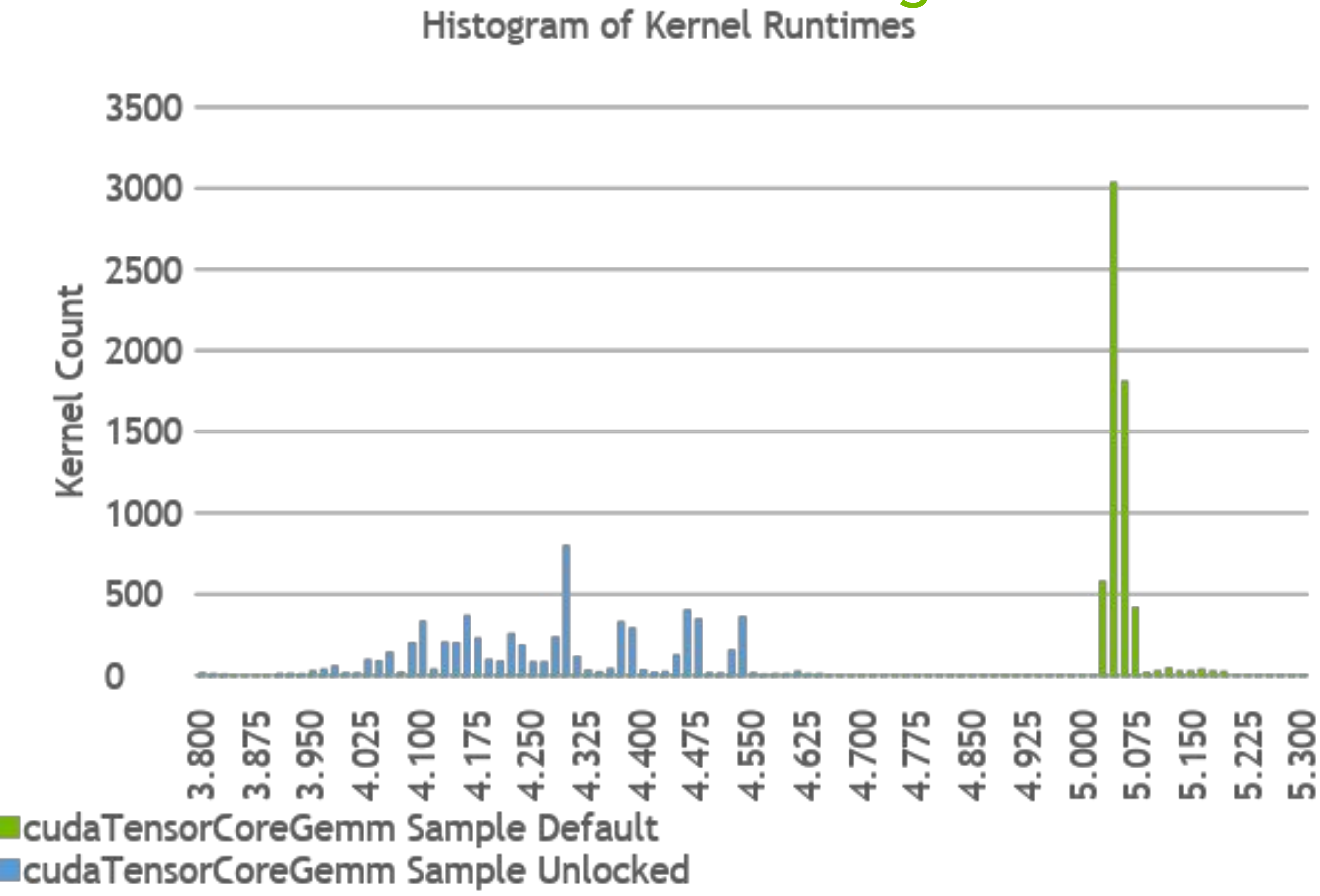
For Volta+

- ‘nvidia-smi –lgc <Default Graphics Clock>’ to lock the GPU clocks regardless of if an application is running

Note that persistence mode must be enabled for the setting to stick

GPU CLOCK SETTINGS

Achieving Stable GPU Benchmarks



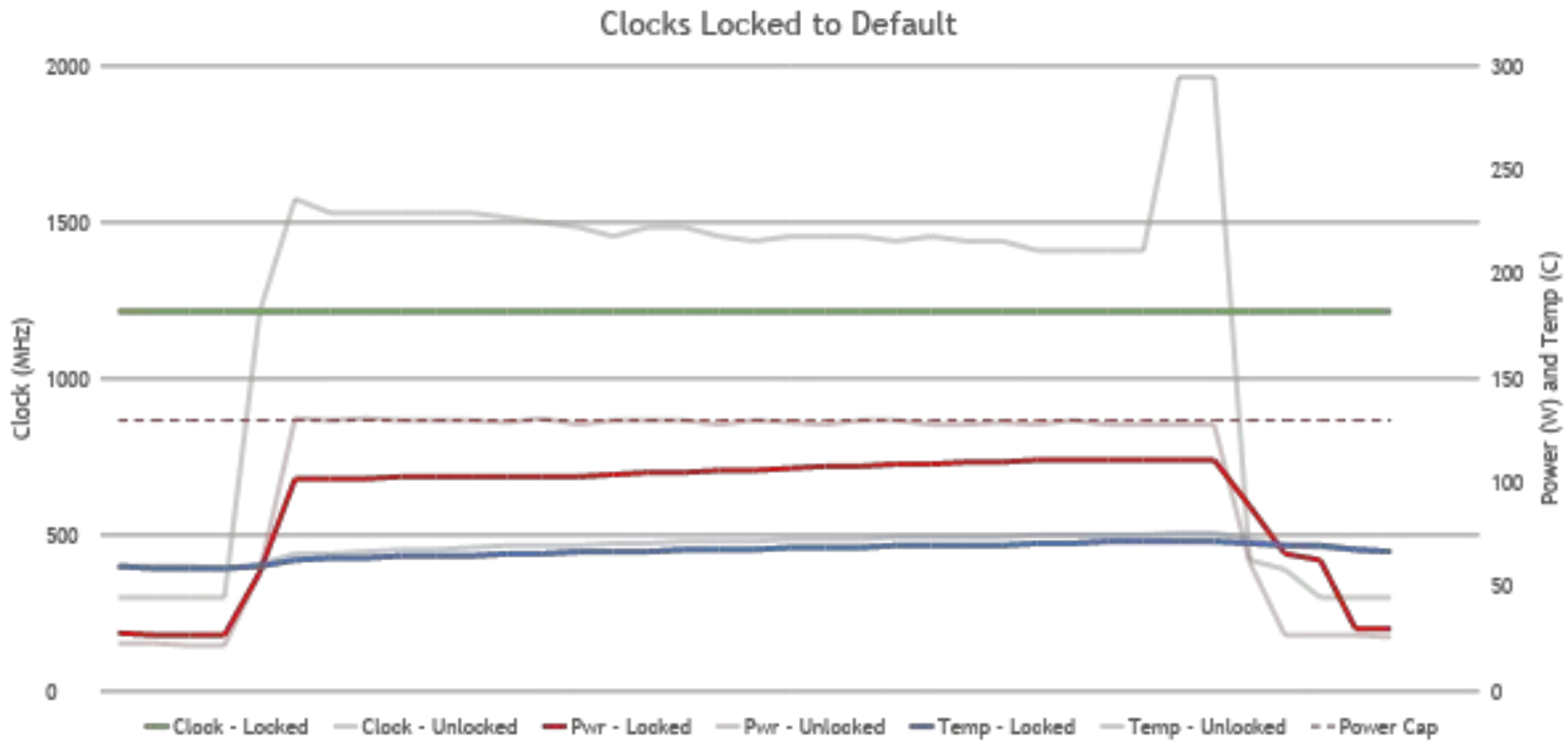
Note that absolute performance may be lower at default clocks, but we're after stable rather than peak performance

Mean Kernel Runtime – 5.05 ms

Relative Standard Deviation – 0.39%

GPU CLOCK SETTINGS

Monitoring Clocks and Throttling



GPU CLOCK SETTINGS

Monitoring Clocks and Throttling

'nvidia-smi dmon' output for both runs

Unlocked:

# gpu	pwr	gtemp	mtemp	sm	mem	enc	dec	mclk	pclk
# Idx	W	C	C	%	%	%	%	MHz	MHz
0	22	59	-	0	0	0	0	405	300
0	57	61	-	1	0	0	0	6500	1215
0	131	66	-	22	12	0	0	6500	1575
0	130	68	-	100	66	0	0	6500	1530
0	130	69	-	100	66	0	0	6500	1530
0	129	70	-	100	65	0	0	6500	1515
0	131	70	-	100	65	0	0	6500	1500
0	128	70	-	100	65	0	0	6500	1485
0	130	71	-	100	64	0	0	6500	1455
0	130	72	-	100	64	0	0	6500	1485
0	128	72	-	100	64	0	0	6500	1455
0	130	72	-	100	63	0	0	6500	1440
0	130	73	-	100	62	0	0	6500	1455
0	130	74	-	100	62	0	0	6500	1440
0	128	74	-	100	62	0	0	6500	1455
0	129	74	-	100	62	0	0	6500	1440
0	130	75	-	100	62	0	0	6500	1410
0	128	75	-	100	62	0	0	6500	1410
0	128	76	-	100	61	0	0	6500	1965
0	128	76	-	60	38	0	0	6500	1965
0	62	73	-	0	0	0	0	6500	420

Locked to Default:

# gpu	pwr	gtemp	mtemp	sm	mem	enc	dec	mclk	pclk
# Idx	W	C	C	%	%	%	%	MHz	MHz
0	27	59	-	0	0	0	0	810	1215
0	57	60	-	0	0	0	0	6500	1215
0	102	63	-	41	22	0	0	6500	1215
0	102	64	-	100	54	0	0	6500	1215
0	102	64	-	100	54	0	0	6500	1215
0	103	67	-	100	54	0	0	6500	1215
0	104	67	-	100	54	0	0	6500	1215
0	105	67	-	100	54	0	0	6500	1215
0	106	68	-	100	54	0	0	6500	1215
0	107	69	-	100	54	0	0	6500	1215
0	108	69	-	100	54	0	0	6500	1215
0	109	70	-	100	54	0	0	6500	1215
0	109	70	-	100	54	0	0	6500	1215
0	110	70	-	100	54	0	0	6500	1215
0	111	71	-	100	54	0	0	6500	1215
0	111	72	-	100	54	0	0	6500	1215
0	111	72	-	100	54	0	0	6500	1215
0	89	71	-	100	54	0	0	6500	1215
0	66	70	-	78	41	0	0	6500	1215
0	63	70	-	0	0	0	0	6500	1215
0	30	68	-	0	0	0	0	810	1215

MEASURING THE RIGHT THING

CUDA JIT COMPILATION

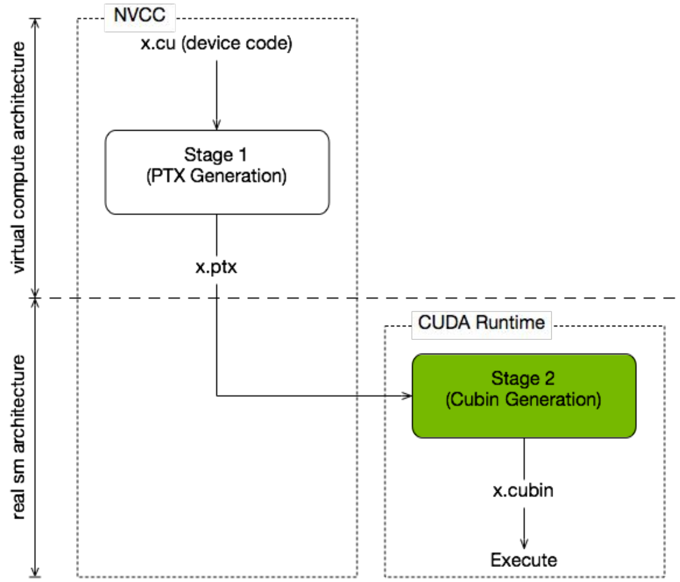
Performance Considerations

When a CUDA fat binary doesn't include code for the architecture to be executed, the PTX (if available) is just-in-time compiled by the driver

In order to reduce CUDA module load time, JIT results are cached on the filesystem

Default locations for JIT cache:

- Linux - ~/.nv/ComputeCache
- Windows - %APPDATA\%NVIDIA\ComputeCache



CUDA JIT COMPILATION

Performance Considerations

For certain environments these default locations can be problematic

- If the location is a network filesystem, access can be slow
- If the location is shared across nodes, concurrent access can result in drops in performance

JIT cache location and usage is configurable

- Environment variable CUDA_CACHE_PATH can be used to set the location
- Environment variable CUDA_CACHE_DISABLE can be used to skip the cache entirely

CUDA JIT COMPILATION

Performance Considerations

JITing can be time consuming, especially on a cache miss

Can be invoked during module load and runtime initialization

Avoid timing JITing when benchmarking code unless specifically required

- Use appropriate architecture flags to create fat binaries to avoid JITing and the JIT cache
- For Example: **-gencode=arch=compute_75,code=sm_75**
- See nvcc documentation for details

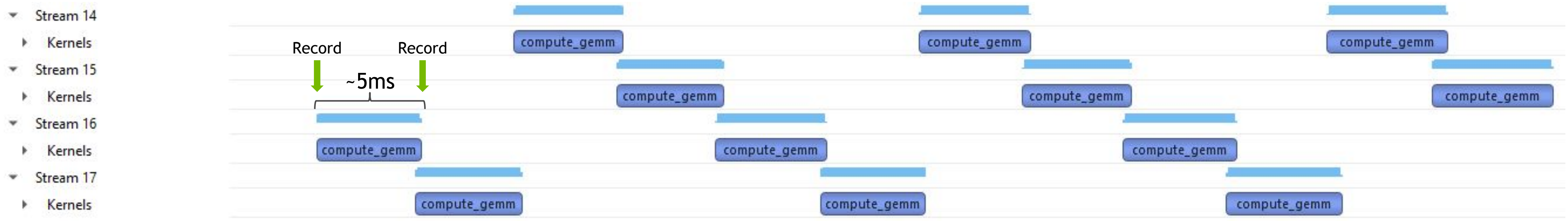
CUDA EVENTS

Timing Event Issues

Using events to time kernels in complex multi-stream cases can result in unexpected results

- Example: Start and end events recorded for each kernel launch across 4 streams

```
for (int i = 0, j = 0; i < NUM_RUNS / NUM_STREAMS; i++) {
    for (int iStream = 0; iStream < NUM_STREAMS; iStream++, j++) {
        cudaEventRecord(startEvents[j], streams[iStream]);
        compute_gemm<<<gridDim, blockDim, SHMEM_SZ, streams[iStream]>>>(…);
        cudaEventRecord(stopEvents[j], streams[iStream]);
    }
}
```



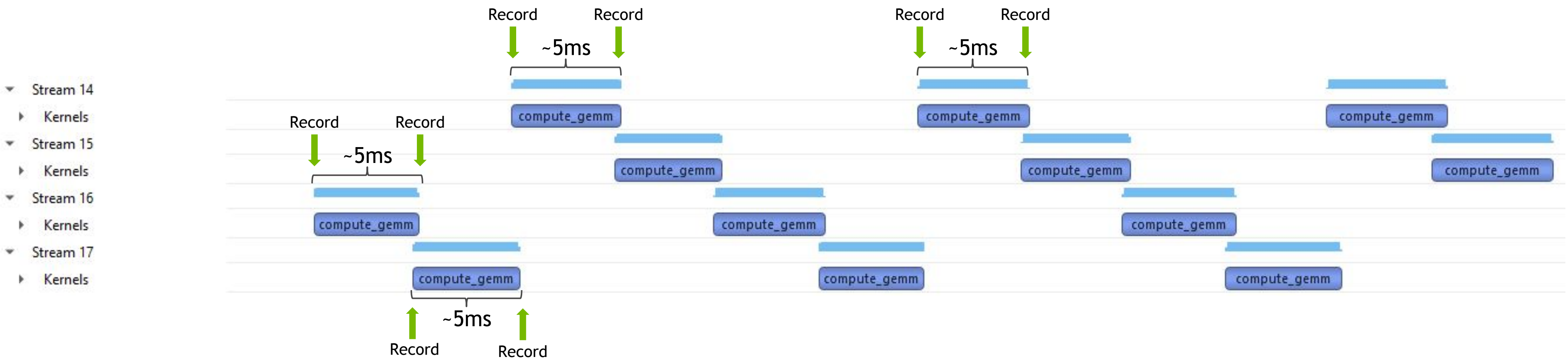
CUDA EVENTS

Timing Event Issues

The expectation might be that each event pair reports ~5ms (the kernel runtime)

- Events have no affinity to the preceding or subsequent GPU work
- Only ordering within the stream is guaranteed

Expected recorded event times:



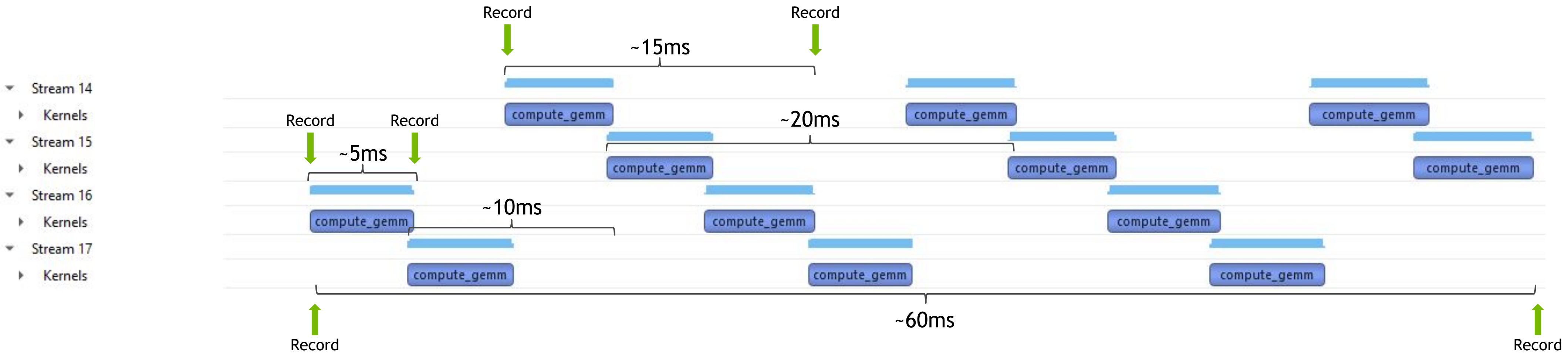
CUDA EVENTS

Timing Event Issues

Start and end events have additional work from other streams interleaved

- Per kernel events report 1-4x actual kernel execution time (with 4 streams)
- Default stream events timing the entire run are accurate

Actual recorded event times:



CUDA EVENTS

Timing Event Issues

Even for single stream, other GPU operations can be executed between the start and end event

Events will record the time the GPU executes the event on the given stream

- Useful for measuring stream work with respect to the CPU
- Useful for coarser measurements, but not short running kernels

Nsight Compute and Nsight Systems are better suited for measuring specific GPU kernels when using multiple streams

- Both have access driver internals that allow for accurate measurement of GPU operations

CUDA EVENTS

Other Event Benchmarking Troubles

Events have timing enabled by default

- Recording a time may result in synchronization, potentially reducing concurrency

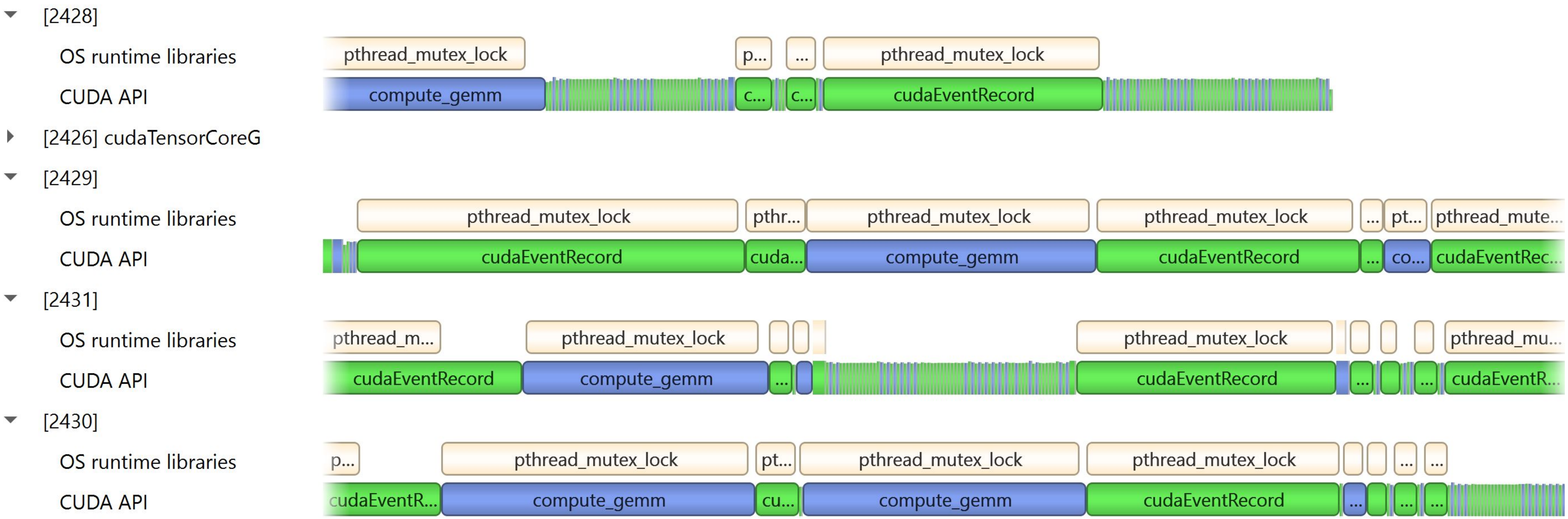
To use events for explicit synchronization or querying, disable timing when creating the event

- Use cudaEventDisableTiming or CU_EVENT_DISABLE_TIMING flags to disable timing on creation

API OVERHEAD

Latency spikes

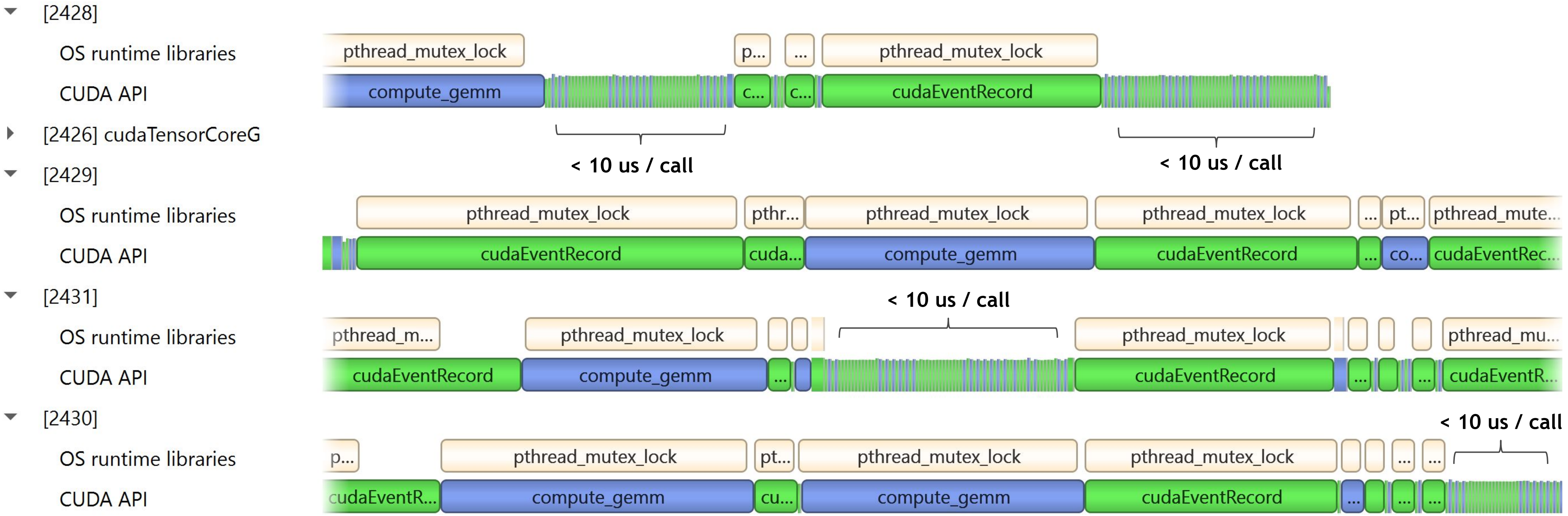
4 threads, 1 stream per thread, loop event record + GEMM + event record in each stream



API OVERHEAD

Latency spikes

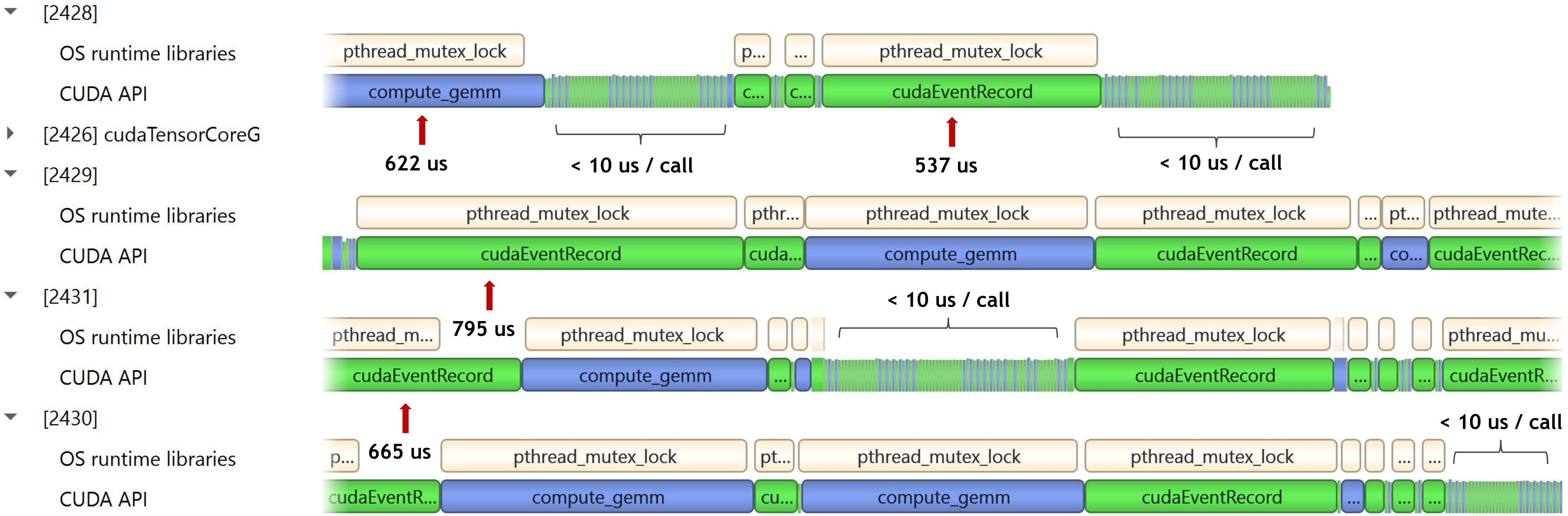
4 threads, 1 stream per thread, loop event record + GEMM + event record in each stream



API OVERHEAD

Latency spikes

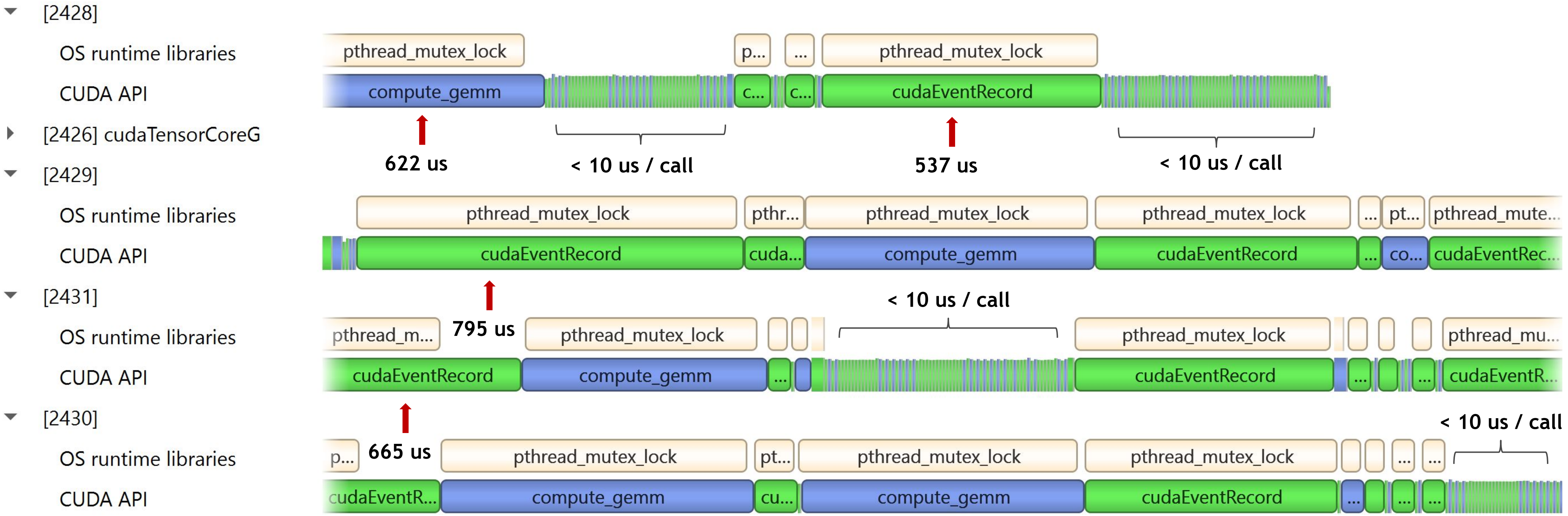
4 threads, 1 stream per thread, loop event record + GEMM + event record in each stream



API OVERHEAD

Lock contention

pthread_mutex is not fair and depends on OS scheduler to select the next thread



API OVERHEAD

How to avoid it?

Approach	Benefit	More information
Submit work from a single CPU worker thread	Eliminates inter-thread lock contention	This presentation
Batch work submission when using many CPU threads	Eliminates some of inter-thread lock contention	GTC 2019 - CE9147 Connect with the Experts: CUDA Platform
Try **CUDA Graphs** to minimize overall API overheads	Reduces overheads by >2x	GTC 2019 - S9240 CUDA: New Features and Beyond
Combine kernels together to avoid API calls	Single kernel eliminates launch and inter-kernel overheads	Cooperative Groups: Flexible CUDA Thread Programming Devblog
Go multi-process with Volta+ MPS	Separates launching threads and avoids locks	GTC 2017 - S7798 Inside Volta

SUMMARY

Best Practices when benchmarking CUDA applications

System stability

- CPU Frequency Scaling – Use performance governor and disable Turbo Boost
- NUMA – Use ‘numactl’ to control NUMA behavior
- GPU clocks – Lock GPU clocks for stable benchmarking

Measuring the right thing

- JIT cache – Check the location or avoid JITing entirely
- CUDA events – Use Nsight tools for better measurements
- API contention – Take steps to avoid lock contention


NVIDIA®