IMPROVE GPU UTILIZATION FROM SYSTEM LEVEL

Click Cheng, NVIDIA Solution Architect

GTC China 2020

WHAT'S ABOUT THE TALK

Welcome

It's

From system level of NVIDIA perspective, proposed several ways to improve GPU utilization;

Discuss several GPU monitoring metrics which reflect real GPU utilization;

Intro each solution mechanism, usage, discuss the benefit in some test cases;

Summary different solution positioning, comparison, etc;

It's Not

Improve GPU utilization from scheduler level;

Optimize GPU utilization from coding level;

OUTLINE

Overview
What's About The Talk
GPU Utilization Discussion

Multi-Process Service
MPS Intro, Usage, Test Cases

Multi-Instance GPU
MIG Intro, Usage, Test Cases

Triton and vGPU Brief
Intro, Test Cases

Quick Summary

OVERVIEW

BACKGROUND

Why Is This Important

GPU is more and more powerful, and more precious.

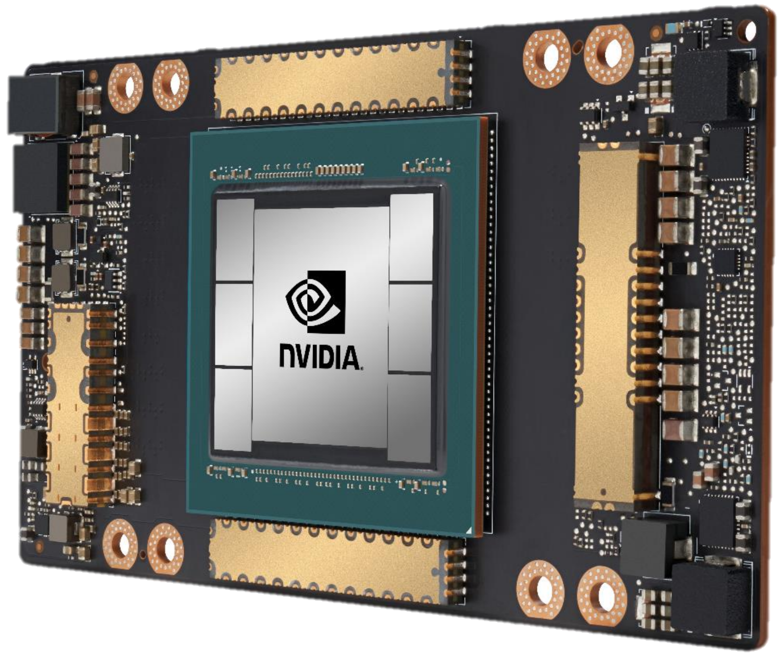
Many applications are benefiting more from more powerful GPU.

While for some lower-utilized application, still can’t fully utilize GPU powerful computing capability.

Example, some developing scenario, inference scenario.

Especially for some inference cases with critical latency limitation, which not allowed batching for inference.

How to share and isolate among processes or users on one GPU?



GPU UTILIZATION

Metrics and Tools

GPU utilization: reflect how busy different resources on GPU are, metrics including GPU core(CUDA core, integer, FP32, Tensor Core), frame buffer(capacity, bandwidth), PCIe RX and TX, NVLink RX and TX, encoder and decoder, etc.

Generally, when we talk about GPU utilization, we are mostly talking about GPU utilization of CUDA core.

GPU utilization reflects an impact on delivered application performance somehow, but not necessarily.

Monitor tools

nvidia-smi or NVML, installed with GPU driver;

DCGM: Data Center GPU Manager, standalone package, using NVML and advanced data center profiling metrics;

GPU UTILIZATION METRIC

From nvidia-smi or NVML

“GPU Utilization” from nvidia-smi or NVML is a rough metric that reflects how busy GPU cores are utilized.

Defined by “Percent of time over the past sample period during which one or more kernels was executing on the GPU”, from NVML API Guide.

Extreme case, the metric is 100% even there’s only one thread launched to run kernel on GPU during past sample period.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.06    Driver Version: 450.51.06    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:06:00.0 Off |                    0 |
| N/A   62C    P0   120W / 300W |  15698MiB / 16160MiB |     78%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A     10210      C   python3                         15693MiB |
+-----------------------------------------------------------------------------+
```

GPU UTILIZATION METRIC

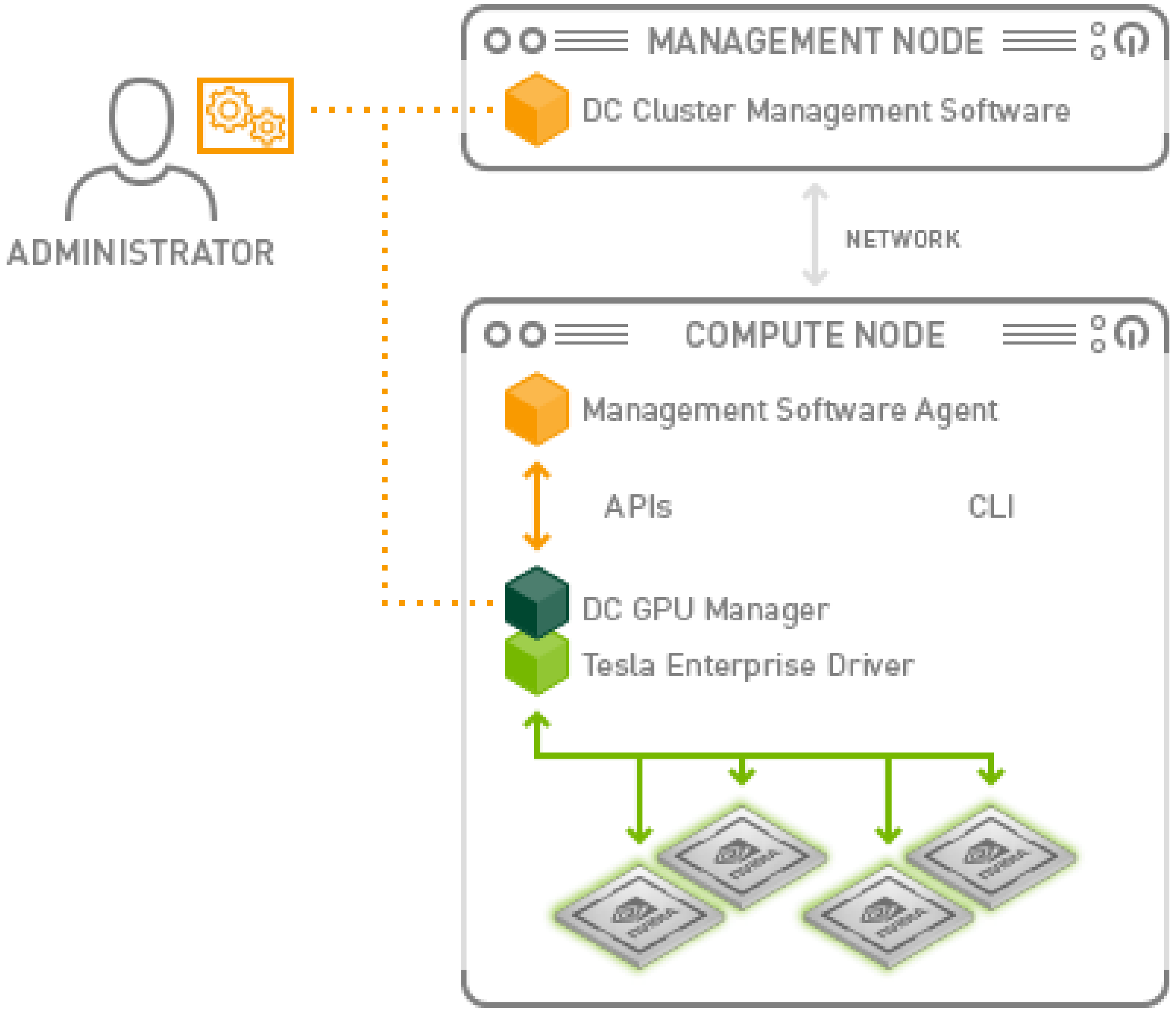
From DCGM

DCGM provides CLI dcgmi and API for C and Python language.

DCGM DCP(Data Center Profiling) provides lower level profiling metrics, which lists several utilization metrics in more accurate.

From these metrics, better reflect how well GPU resources are utilized to some extent.

Well, one GPU has many different resources(computing, memory, IO), it’s highly recommended to capture several metrics to understand GPU utilization, not just one or two.



GPU UTILIZATION METRIC

DCGM DCP Metrics

Metric	Definition	DCGM Field ID
Graphics Engine Activity	Ratio of time the graphics engine is active. The graphics engine is active if a graphics/compute context is bound and the graphics pipe or compute pipe is busy.	DCGM_FI_PROF_GR_ENGINE_ACTIVE
SM Activity	The ratio of cycles an SM has at least 1 warp assigned (computed from the number of cycles and elapsed cycles)	DCGM_FI_PROF_SM_ACTIVE
SM Occupancy	The ratio of number of warps resident on an SM. (number of resident as a percentage of the theoretical maximum number of warps per elapsed cycle)	DCGM_FI_PROF_SM_OCCUPANCY
Tensor Utilization	The ratio of cycles the tensor (HMMA) pipe is active (off the peak sustained elapsed cycles)	DCGM_FI_PROF_PIPE_TENSOR_ACTIVE
Memory BW Utilization	The ratio of cycles the device memory interface is active sending or receiving data.	DCGM_FI_PROF_DRAM_ACTIVE
FLOP Counts	Ratio of cycles the fp64 /fp32 / fp16 / HMMA|IMMA pipes are active.	DCGM_FI_PROF_PIPE_FPXY_ACTIVE
NVLink Utilization	The number of bytes of active NVLink rx or tx data including both header and payload.	DCGM_FI_DEV_NVLINK_BANDWIDTH_L0
PCIe Utilization	pci__bytes_{rx, tx} - The number of bytes of active pcie rx or tx data including both header and payload.	DCGM_FI_PROF_PCIE_[T|R]X_BYTES

GPU UTILIZATION METRIC

Using dcgmi

Recommended monitor command with dcgmi

$ dcgmi dmon -e 1001,1002,1004,1005,1009,1010,1011,1012,150,155,110,111

```
dgxuser@a100:~$ dcgmi dmon -e 1001,1002,1004,1005,1009,1010,1011,1012,150,155,110,111
# Entity  GRACT  SMACT  TENSO  DRAMA              PCITX              PCIRX              NVLTX              NVLRX  TMPTR      POWER  SACLK  MACLK
    Id                                                                                                            C          W
    GPU 0  0.931  0.777  0.175  0.496          175899291         1532954951         1547634958         1553333956     52    323.689   1410   1215
    GPU 1  0.948  0.780  0.173  0.496          172945598         1507859117         1522127704         1522126460     50    213.963   1410   1215
    GPU 2  0.952  0.778  0.175  0.493          178507418         1557783818         1572668487         1572504828     48    359.610   1410   1215
    GPU 3  0.962  0.793  0.178  0.503          164054321         1428701446         1327745638         1327396166     52    226.107   1410   1215
    GPU 4  0.960  0.786  0.179  0.499          163908021         1430858946         1288201051         1287639531     64    392.270   1410   1215
    GPU 5  0.952  0.797  0.182  0.506          182644334         1599554874         1235853101         1233988554     62    341.524   1410   1215
    GPU 6  0.966  0.817  0.200  0.508          132741767         1148660264         1129637355         1127111684     64    258.063   1410   1215
    GPU 7  0.999  0.867  0.325  0.451            8908656           34245363                  0                  0     67    380.955   1410   1215
    GPU 0  0.950  0.793  0.179  0.505          162992146         1418772939         1429455794         1435422194     54    304.839   1410   1215
    GPU 1  0.954  0.793  0.179  0.505          162944796         1418947251         1430185344         1430185344     52    201.105   1410   1215
    GPU 2  0.959  0.795  0.179  0.505          162966713         1419469072         1430752928         1430665363     53    372.072   1410   1215
    GPU 3  0.962  0.796  0.179  0.505          162992814         1418956709         1430003326         1429872315     56    195.564   1410   1215
    GPU 4  0.960  0.792  0.179  0.505          162681409         1418483393         1431427800         1430779751     66    400.533   1410   1215
    GPU 5  0.948  0.789  0.179  0.506          162794813         1419172557         1435095820         1431911846     65    355.586   1410   1215
    GPU 6  0.957  0.794  0.179  0.505          162844371         1418843705         1439494242         1434260115     67    292.070   1410   1215
    GPU 7  0.958  0.797  0.180  0.506          163341225         1422783028         1440327549         1443650284     70    384.132   1410   1215
    GPU 0  0.949  0.793  0.179  0.505          163030005         1419242144         1431636763         1440810765     55    237.659   1410   1215
    GPU 1  0.954  0.793  0.179  0.505          162773503         1418681427         1431965210         1431965210     53    184.241   1410   1215
    GPU 2  0.957  0.795  0.179  0.506          162881890         1419208015         1432242919         1432242919     53    366.301   1410   1215
    GPU 3  0.959  0.797  0.179  0.506          163018599         1419626682         1432350659         1432350659     56    225.281   1410   1215
    GPU 4  0.957  0.792  0.180  0.506          162612068         1418432187         1433020167         1432423976     66    396.763   1410   1215
    GPU 5  0.949  0.790  0.179  0.506          162527136         1417457784         1436326347         1431757595     65    278.087   1410   1215
    GPU 6  0.956  0.794  0.179  0.505          162874341         1419286866         1446253398         1437670082     68    330.311   1410   1215
    GPU 7  0.957  0.797  0.181  0.506          162804092         1419116905         1441752283         1446325537     71    402.675   1410   1215
    GPU 0  0.951  0.796  0.180  0.507          163049364         1419855949         1431014322         1437445669     55    178.524   1410   1215
    GPU 1  0.951  0.795  0.180  0.507          162991159         1420276463         1431002457         1431002457     52    237.659   1410   1215
    GPU 2  0.958  0.797  0.180  0.507          162890127         1419226321         1430767359         1430767359     53    366.837   1410   1215
    GPU 3  0.960  0.797  0.179  0.506          162951049         1419130718         1430934017         1430371626     56    320.427   1410   1215
    GPU 4  0.957  0.794  0.181  0.506          162632607         1418105919         1431840408         1430961031     67    387.907   1410   1215
    GPU 5  0.948  0.791  0.180  0.507          162692025         1418504737         1435486856         1431624416     66    202.787   1410   1215
    GPU 6  0.958  0.795  0.180  0.507          162784709         1418531037         1441021899         1435473584     68    383.275   1410   1215
    GPU 7  0.955  0.798  0.182  0.507          162806309         1418604281         1436151038         1440574380     71    408.494   1410   1215
    GPU 0  0.954  0.795  0.180  0.506          162990387         1418928837         1429538441         1434605453     55    231.030   1410   1215
    GPU 1  0.953  0.794  0.179  0.506          162726406         1418134181         1429561807         1429561807     53    327.149   1410   1215
    GPU 2  0.957  0.795  0.179  0.506          162749917         1418119150         1429417688         1429417688     54    318.781   1410   1215
    GPU 3  0.960  0.798  0.179  0.506          162980413         1418831034         1429747257         1429747257     56    366.668   1410   1215
    GPU 4  0.958  0.794  0.181  0.507          162550382         1417626219         1430946788         1429692646     67    355.460   1410   1215
    GPU 5  0.948  0.790  0.179  0.506          162619862         1417919747         1434448412         1431062748     65    225.026   1410   1215
    GPU 6  0.958  0.796  0.180  0.506          162713169         1418220725         1438159351         1434345897     68    394.858   1410   1215
    GPU 7  0.956  0.797  0.181  0.506          162676691         1417962711         1434673417         1438058602     72    403.272   1410   1215
```

MULTI-PROCESS SERVICE

HYPER QUEUE

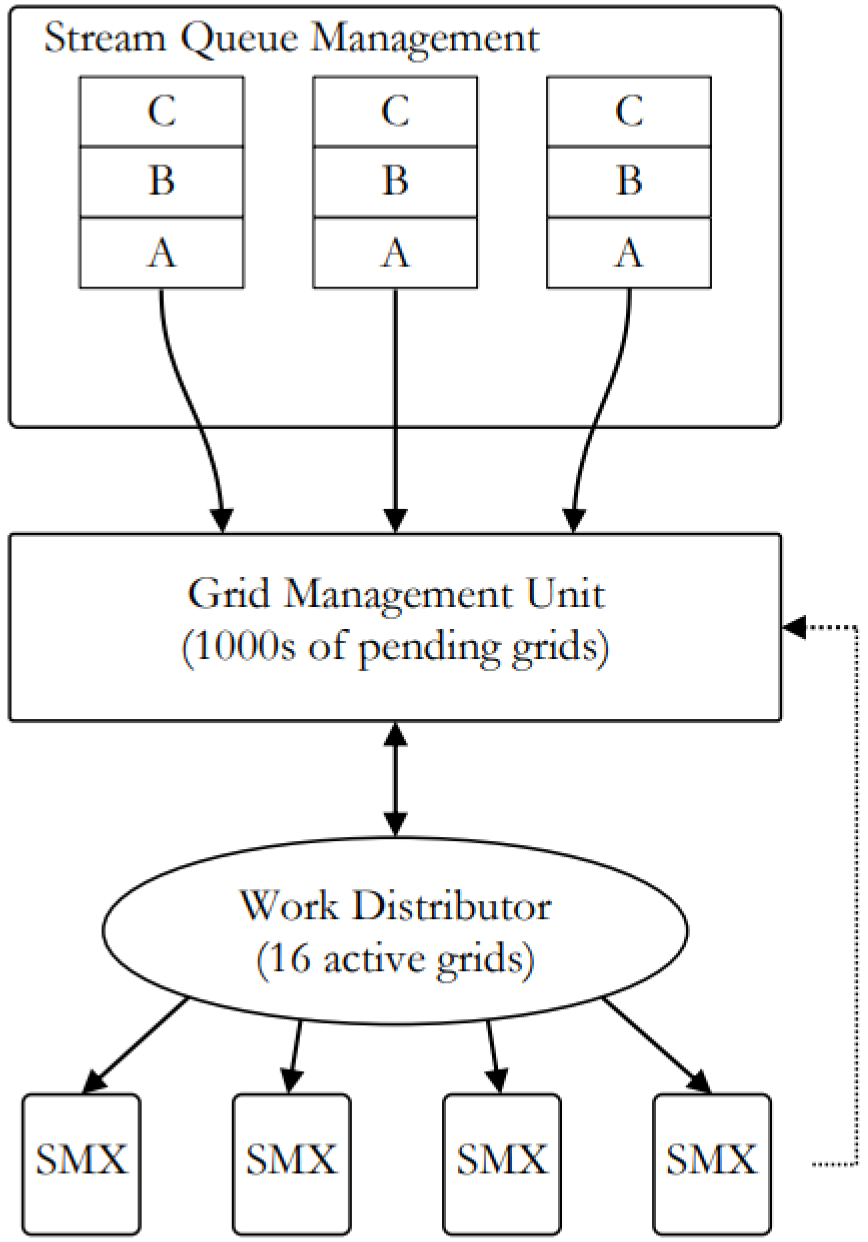
Behind MPS

Hyper-Q is introduced since Kepler GPU.

To enable multiple CPU threads or processes to launch work on a single GPU simultaneously.

Supported connection types:

- Multiple CUDA streams;
- Multiple CPU threads;
- Multiple CPU processes;

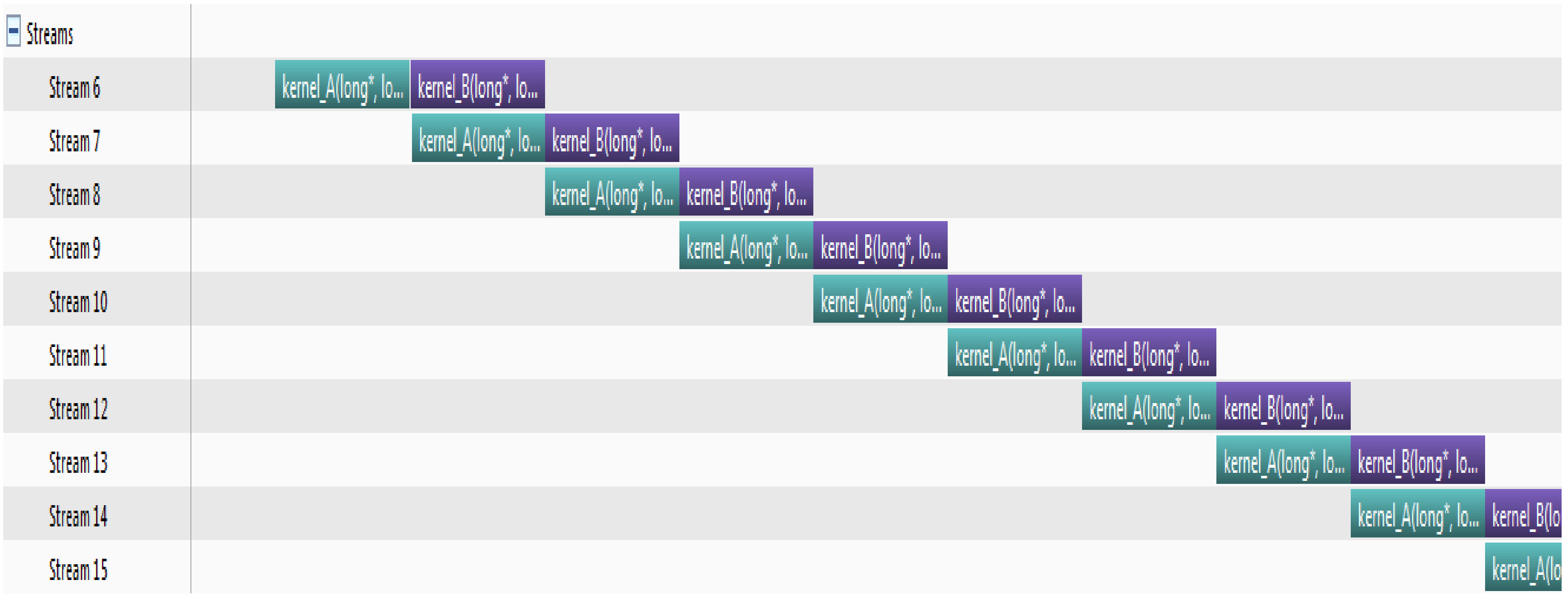


Hyper-Q whitepaper:
https://developer.download.nvidia.com/compute/DevZone/C/html_x64/6_Advanced/simpleHyperQ/doc/HyperQ.pdf

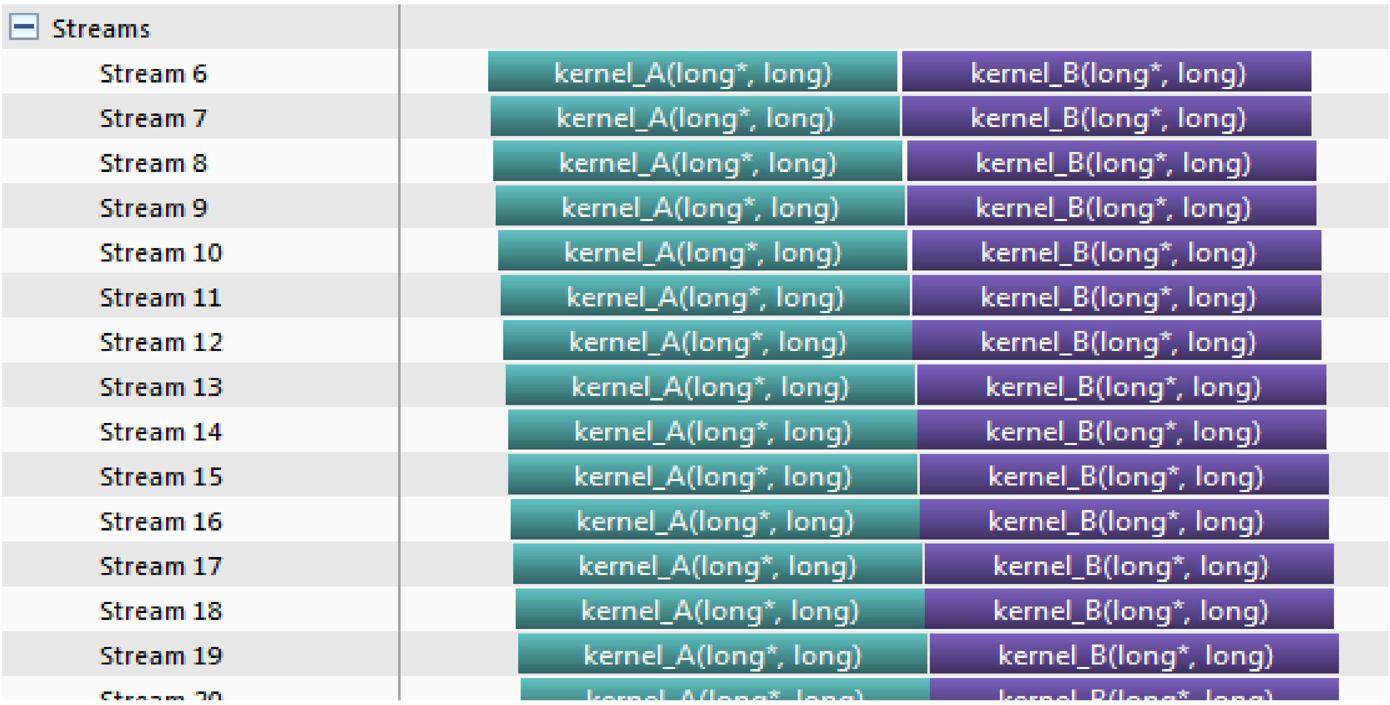
HYPER QUEUE

Example: $CUDA_PATH/samples/6_Advanced/simpleHyperQ

```
for (int i = 0 ; i < nstreams ; ++i)
{
kernel_A<<<1,1,0,streams[i]>>>(&d_a[2*i], time_clocks);
total_clocks += time_clocks;
kernel_B<<<1,1,0,streams[i]>>>(&d_a[2*i+1], time_clocks);
total_clocks += time_clocks;
}
```



Device without Hyper-Q



Device with Hyper-Q

MULTI-PROCESS SERVICE

What's MPS

An alternative, binary-compatible implementation of the CUDA Application Programming Interface (API).

Based on GPU Hyper-Q capability

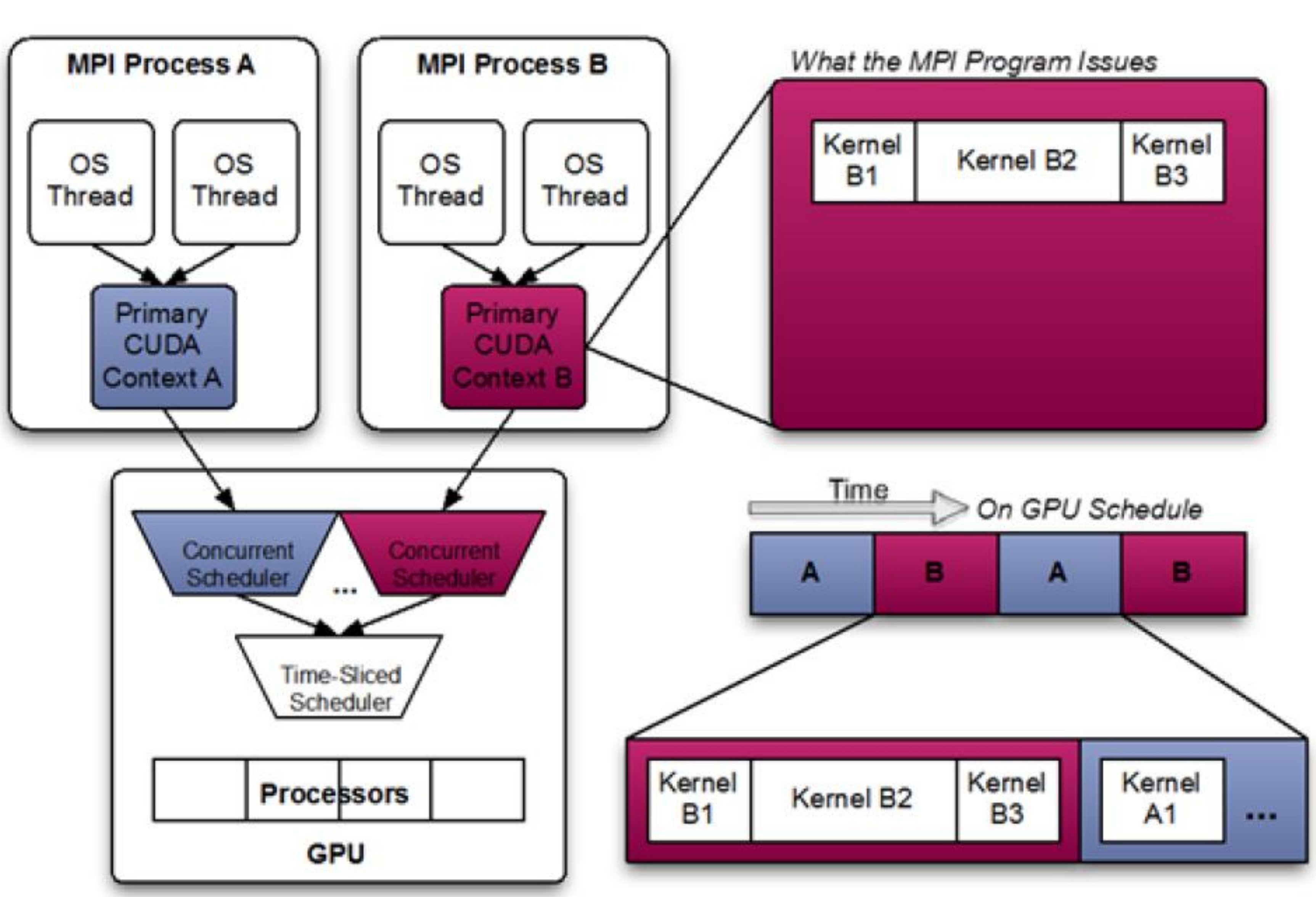
- Enabling multiple CPU processes sharing one GPU context;
- Allowing kernels and memcpy in different processes can be executed simultaneously on the same GPU, to utilize GPU better;

MPS includes

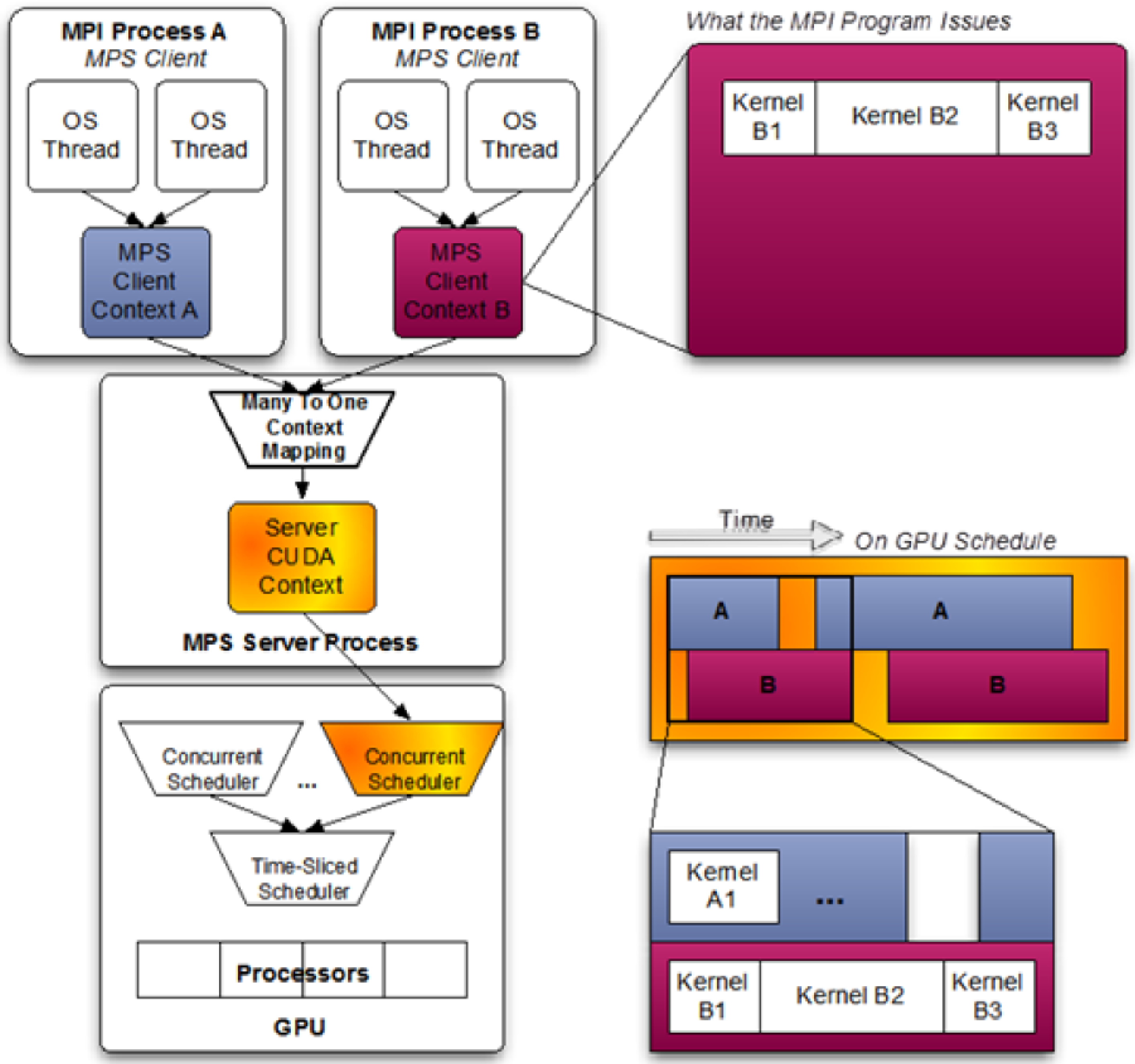
- Control Daemon Process – The control daemon is responsible for starting and stopping the server, as well as coordinating connections between clients and servers.
- Server Process – The server is the clients' shared connection to the GPU and provides concurrency between clients.
- Client Runtime – The MPS client runtime is built into the CUDA Driver library and may be used transparently by any CUDA application.

MULTI-PROCESS SERVICE

Without MPS VS With MPS



Without MPS



With MPS

MULTI-PROCESS SERVICE

MPS Architecture

System-wide provisioning with multiple users.

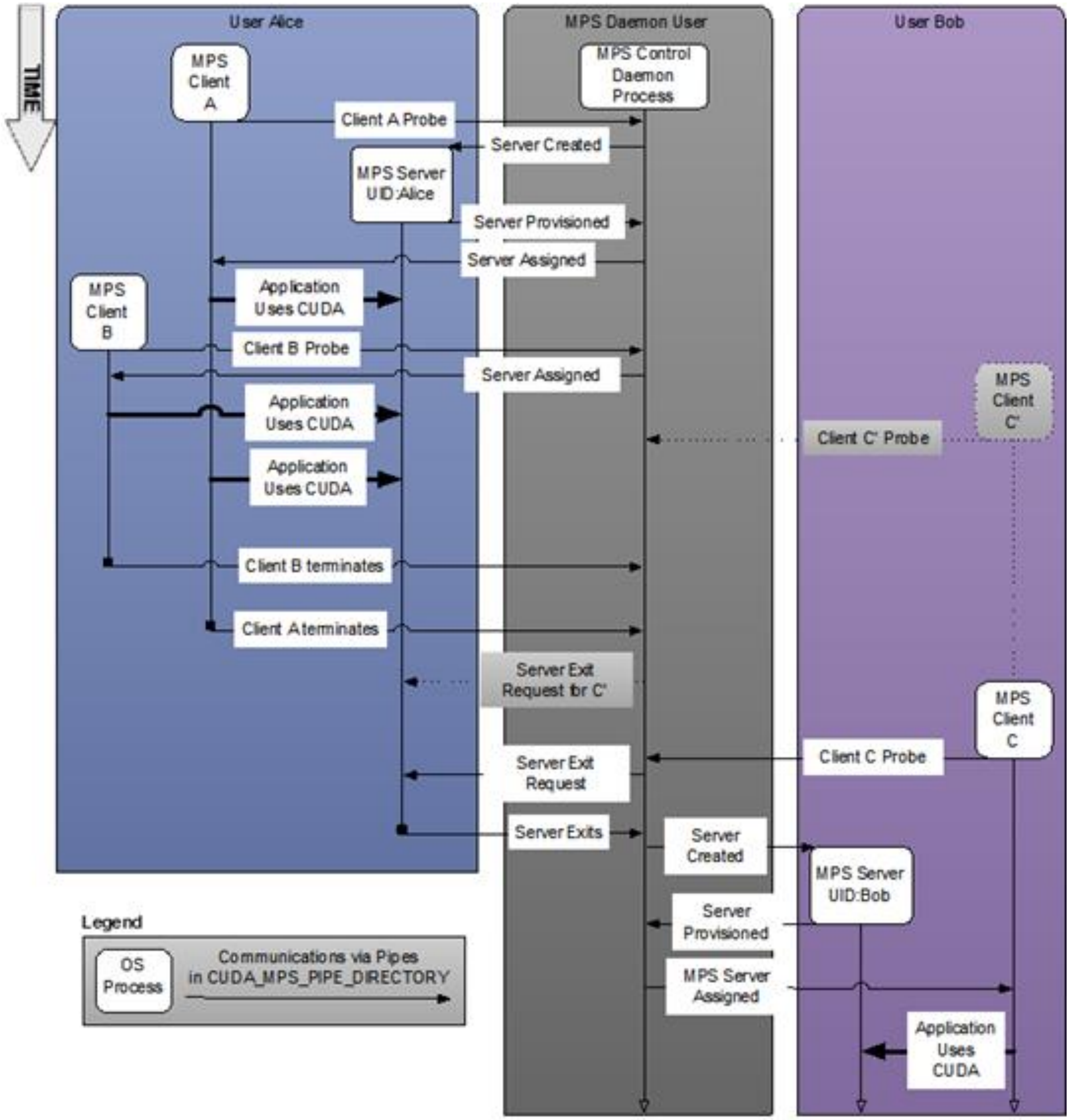
Client A from User 1 request;

Daemon create MPS server for User 1 and Client A runs;

Client B from User 1 request and assigned to MPS server, and to run;

Client C from User 2 request, and pending;

Util all clients from User 1 running end and MPS server exit for User 1, Daemon create MPS server for User 2, and Client C begin to run;



MULTI-PROCESS SERVICE

MPS Benefits

GPU Utilization

A single process may not utilize all the compute and memory-bandwidth capacity available on the GPU. MPS allows kernel and memcopy operations from different processes to overlap on the GPU, achieving higher utilization and shorter running times.

Reduced on-GPU Context Storage

The MPS server allocates one copy of GPU storage and scheduling resources shared by all its clients, thus reduces the resource storage.

Reduced on-GPU Context Switching

The MPS server shares one set of scheduling resources between all of its clients, eliminating the overhead of swapping when the GPU is scheduling between those clients.

MULTI-PROCESS SERVICE

Potential Applications for MPS

Application process does not generate enough work to saturate the GPU. Applications like this are identified by having a small number of blocks-per-grid.

Application shows a low GPU occupancy because of a small number of threads-per-grid.

In strong-scaling case, some MPI processes may underutilize the available compute capacity.

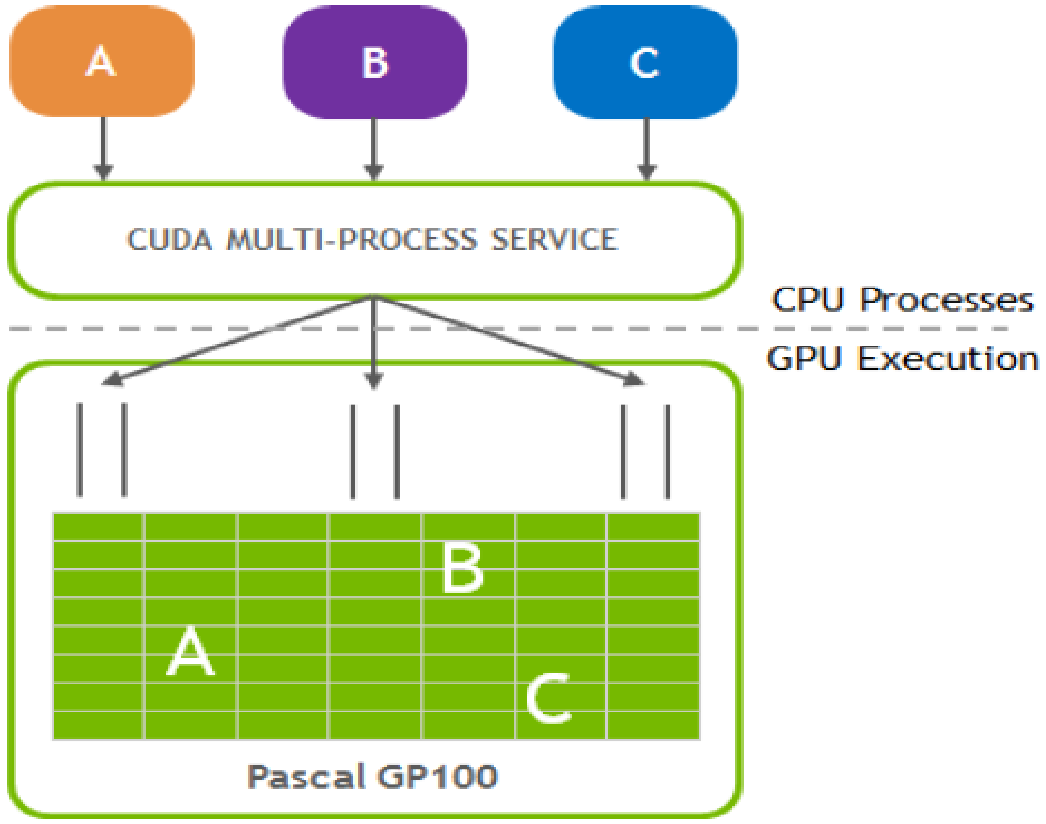
Especially for AI inference, with critical latency limitation, which not allowed batching for inference.

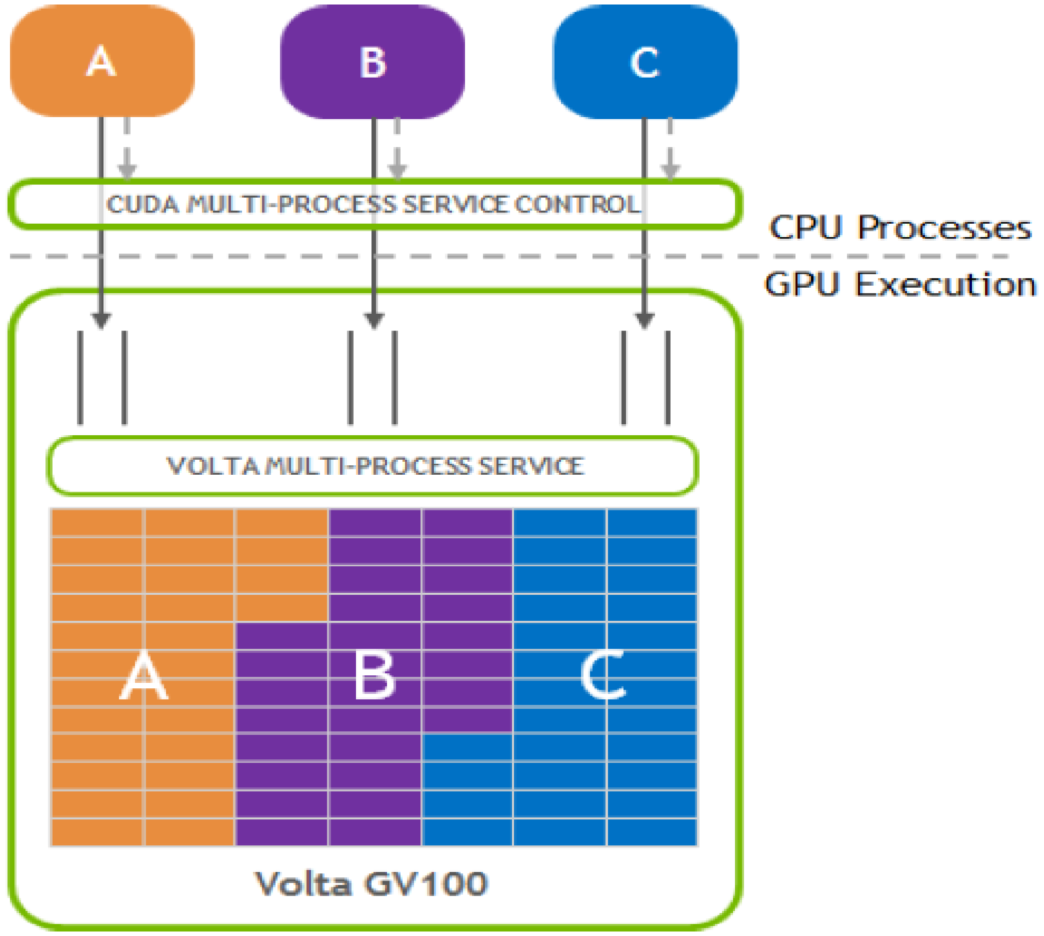
MULTI-PROCESS SERVICE

Volta MPS

Volta MPS provides a few key improvements, compared with pre-Volta:

- Volta MPS clients submit work directly to the GPU without passing through the MPS server.
- Each Volta MPS client owns its own GPU address space instead of sharing GPU address space with all other MPS clients.
- Volta MPS supports limited execution resource provisioning for Quality of Service (QoS).





MULTI-PROCESS SERVICE

MPS Usage

Start MPS daemon process

nvidia-cuda-mps-control –d

Check MPS process

ps -ef | grep mps

Recommend to set compute mode to exclusive

sudo nvidia-smi -c EXCLUSIVE_PROCESS

Quit MPS daemon

echo quit | nvidia-cuda-mps-control

MULTI-PROCESS SERVICE

MPS Usage

nvidia-smi shows when running eight trtexec processes with MPS:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.67    Driver Version:418.67    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:06:00.0 Off |                  Off |
| N/A   46C    P0   140W / 300W |   7027MiB / 16160MiB |    100%      Default |
+-------------------------------+----------------------+----------------------+


+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     81016      C   nvidia-cuda-mps-server                        29MiB |
|    0     81074      C   trtexec                                      873MiB |
|    0     81075      C   trtexec                                      873MiB |
|    0     81076      C   trtexec                                      873MiB |
|    0     81077      C   trtexec                                      873MiB |
|    0     81078      C   trtexec                                      873MiB |
|    0     81079      C   trtexec                                      873MiB |
|    0     81080      C   trtexec                                      873MiB |
|    0     81081      C   trtexec                                      873MiB |
+-----------------------------------------------------------------------------+
```

MPS TEST CASE 1

Simple Kernel with One Thread Running

Simple kernel code: (Ignore the computing content)

```
__global__ void testMaxFlopsKernel(float * pData, int nRepeats, float v1, float v2)
{
int tid = blockIdx.x* blockDim.x+ threadIdx.x;
float s = pData[tid], s2 = 10.0f - s, s3 = 9.0f - s, s4 = 9.0f - s2;
for(int i = 0; i < nRepeats; i++)
{
s=v1-s*v2;
}
pData[tid] = ((s+s2)+(s3+s4));
}
```

To test: run four processes with and without MPS

To profile: profiling analysis the running characteristic

MPS TEST CASE 1

Test Results

Run multiple processes with mpirun, command like: mpirun –np $NP ./testMPS

Category	Average Wall Clock Time		
	1 Process	2 Processes	4 Processes
MPS OFF	2924 ms	6013 ms	12002 ms
MPS ON	2924 ms	2924 ms	2924 ms

Without MPS, the kernel running time increases linearly along with the number of processes.

With MPS, the kernel run time of multi processes is almost the same as one process.

This is the extreme case, but it’s the best case to show MPS benefit.

MPS TEST CASE 1

Profiling Analysis

Use nvprof to capture trace:

```
node1:~$ nvprof -o ./profile-test2-%p --profile-child-processes mpirun -np 2 ./testMPS
==56763== NVPROF is profiling process 56763, command: ./testMPS
==56768== NVPROF is profiling process 56768, command: ./testMPS
…
Rank0: BlockSize(1, 1, 1), GirdSize(1, 1, 1)
Rank0: Iteration: 1, Total Elapsed Time: 2918.924ms, Single kernel cost time: 2918.924ms
Rank0: Performance: 0.685GFLOPS
Rank1: BlockSize(1, 1, 1), GirdSize(1, 1, 1)
Rank1: Iteration: 1, Total Elapsed Time: 2917.827ms, Single kernel cost time: 2917.827ms
Rank1: Performance: 0.685GFLOPS
…
==56768== Generated result file: /home/dgx/src/testMPS/profile-test2-56768
…
==56763== Generated result file: /home/dgx/src/testMPS/profile-test2-56763
```

Then import into NVVP profiler tool for visual profiling analysis.

MPS TEST CASE 1

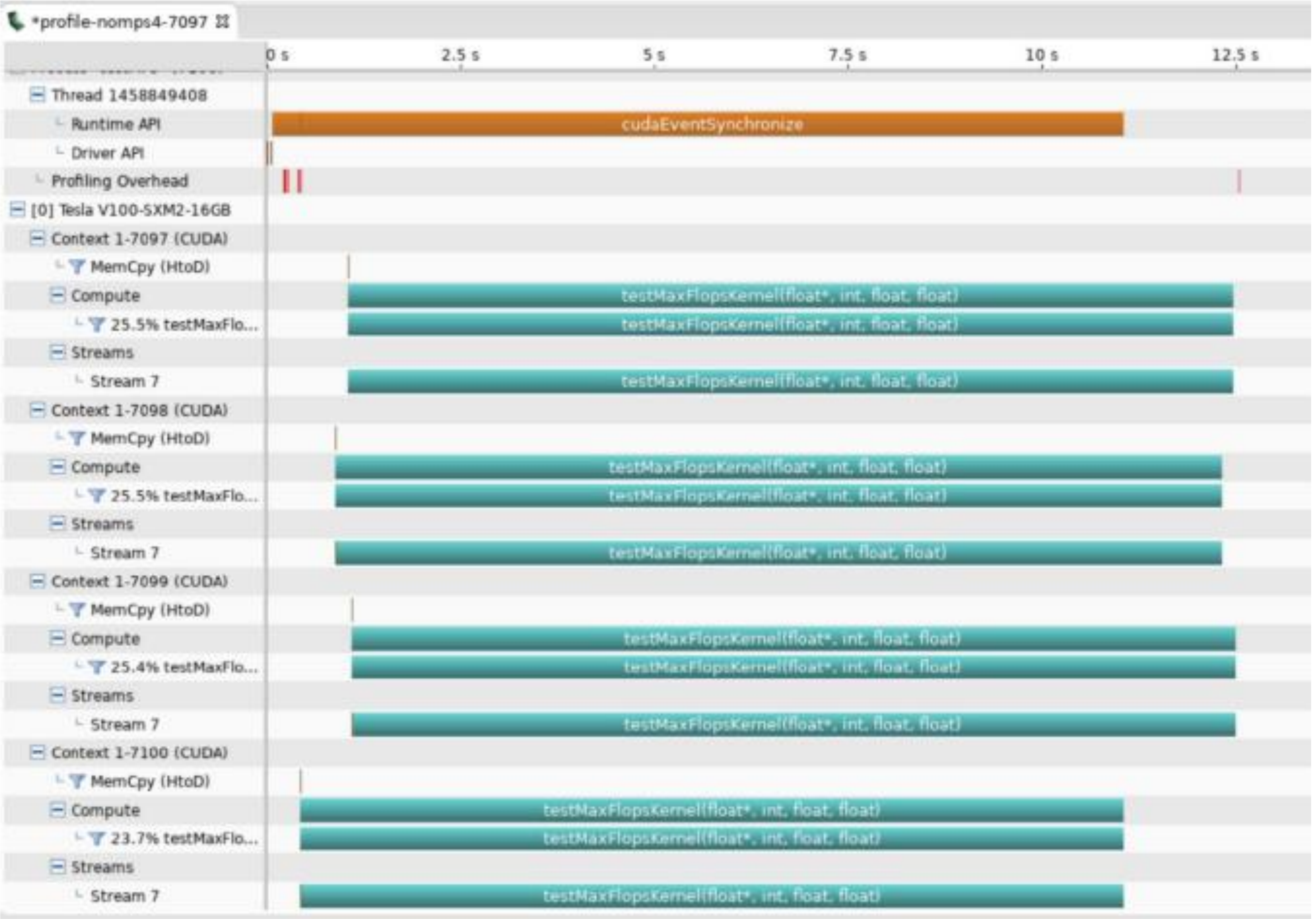
Profiling Analysis: Without MPS

Without MPS, four processes.

Four CUDA contexts on a V100 GPU.

Although it seems like that they are running concurrently, the execution time for each kernel is lengthened.

That is because that they are running under the GPU time slice rotation scheduling mechanism. These CUDA contexts need to be switched in each time slice which introduces extra time overhead.



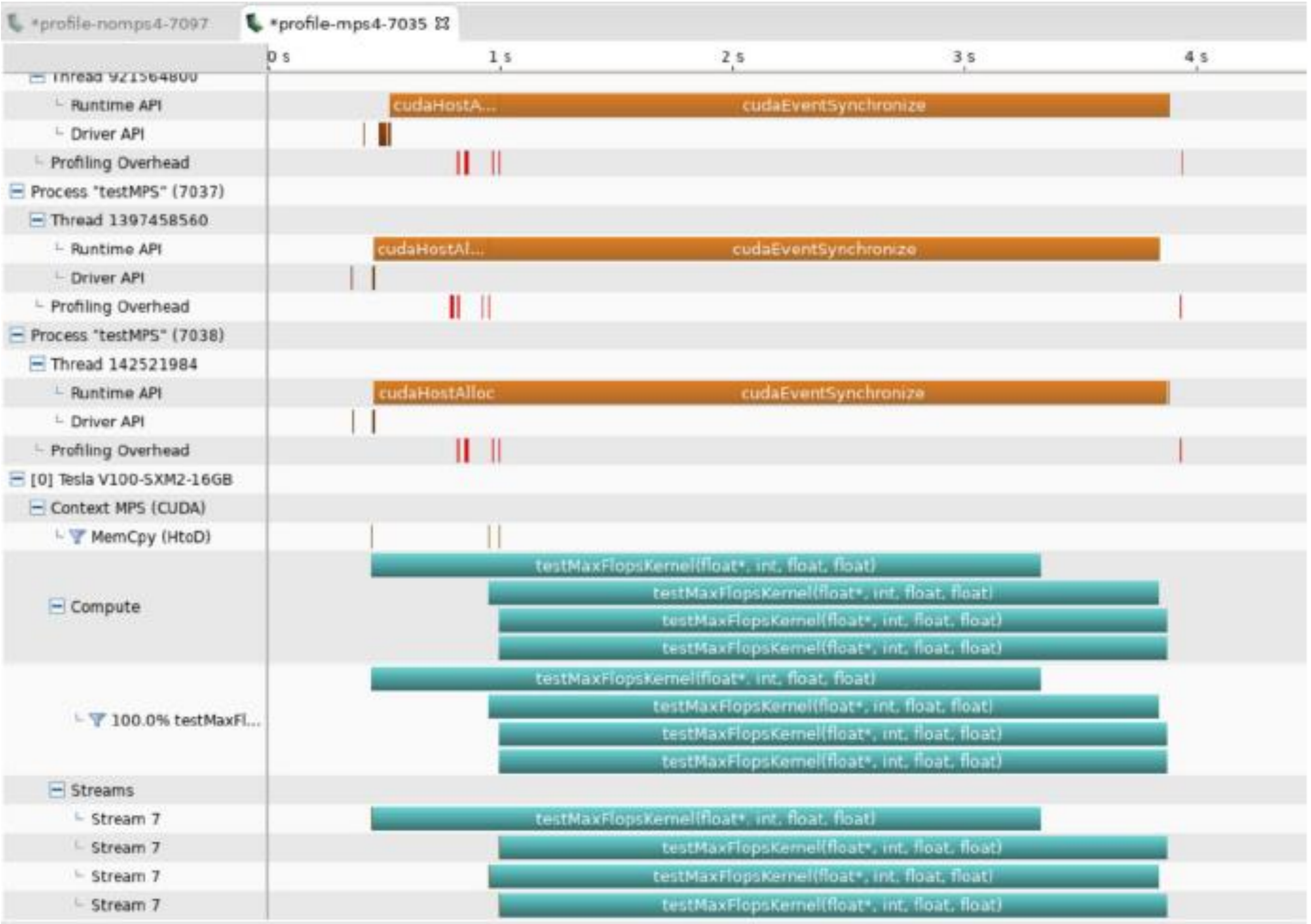
MPS TEST CASE 1

Profiling Analysis: With MPS

With MPS, four processes.

Only one CUDA context to run these four processes.

The kernels from different processes are really running overlapped.



MPS TEST CASE 2

ResNet-50 Inference in 7ms Budget

This example is to run ResNet-50 inference with TensorRT engine.

We use NGC container “nvcr.io/nvidia/tensorrt:19.07-py3” on SXM2 V100 16GB.

We run and compare several scenarios in 7ms inference time budget:

- Batching in single process;
- No batching(batch size is 1) in multiple processes, without MPS;
- No batching(batch size is 1) in multiple processes, with MPS;
- Batching and multiple processes combination;

At the same time, we capture some utilization metrics with dcgmi, to quantify GPU usage.

dcgmi dmon -e 1001,1002,1004,1005,1009,1010,1011,1012

MPS TEST CASE 2

Steps to Test

Start container

nvidia-docker run -it --name click-trt --privileged -v /home/click/models/:/click nvcr.io/nvidia/tensorrt:19.07-py3 bash

Build out ResNet-50 TRT engine (using caffemodel here)

Example, for batch size 1, 32, …

trtexec --batch=1 --iterations=100 --workspace=1024 --deploy=/click/ResNet-50-deploy.prototxt --model=/click/ResNet-50-model.caffemodel --output=prob --fp16 --saveEngine=/workspace/rn50-bs1.engine

trtexec --batch=32 --iterations=100 --workspace=1024 --deploy=/click/ResNet-50-deploy.prototxt --model=/click/ResNet-50-model.caffemodel --output=prob --fp16 --saveEngine=/workspace/rn50-bs32.engine

Test in single process

trtexec --loadEngine=/workspace/rn50-bs1.engine --iterations=1000 --workspace=1024 --fp16

trtexec --loadEngine=/workspace/rn50-bs32.engine --iterations=10000 --workspace=1024 --fp16 --batch=32

Test in multi processes with MPI

mpirun -np 8 --allow-run-as-root trtexec --loadEngine=/workspace/rn50-bs1.engine --iterations=1000 --workspace=1024 --fp16 > trt-mps-mpi-8.log

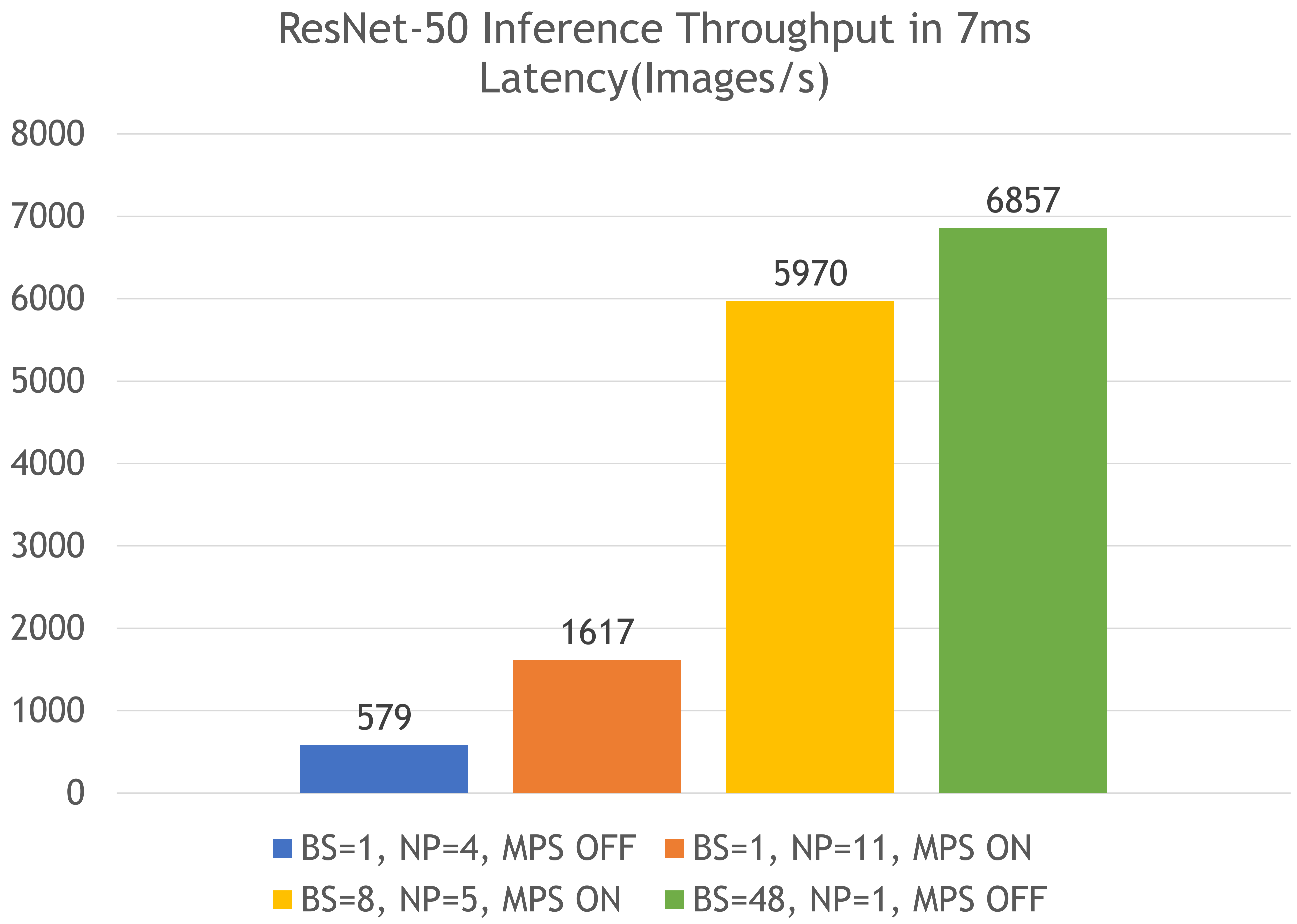
MPS TEST CASE 2

Test Results

Batching is the recommended way to reach best throughput.

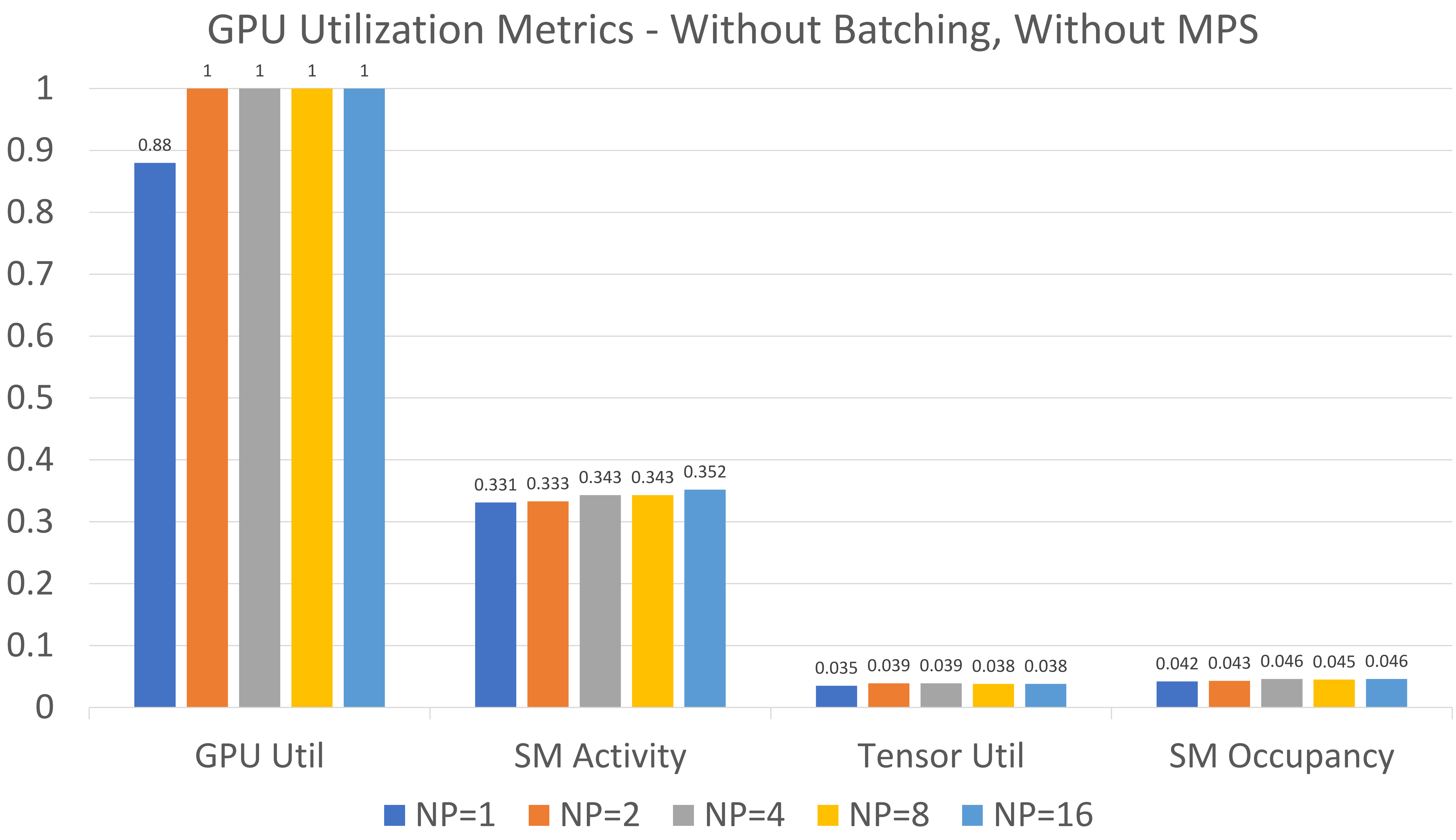
Without batching, i.e. BS=1 cases, MPS can bring ~3X throughput.

Batching and MPS can be combined, to improve throughput to some extent.



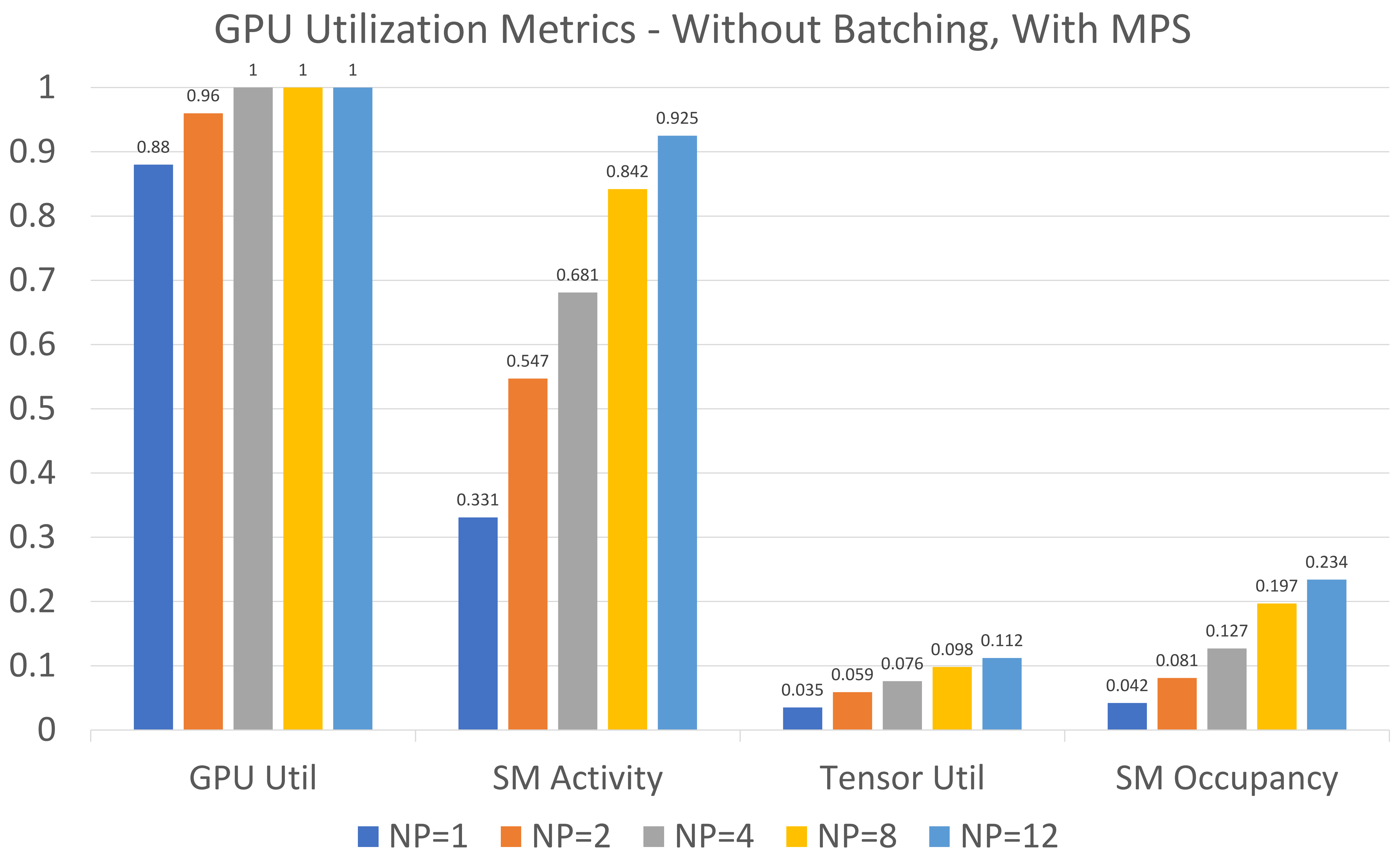
MPS TEST CASE 2

GPU Utilization Metrics – MPS OFF



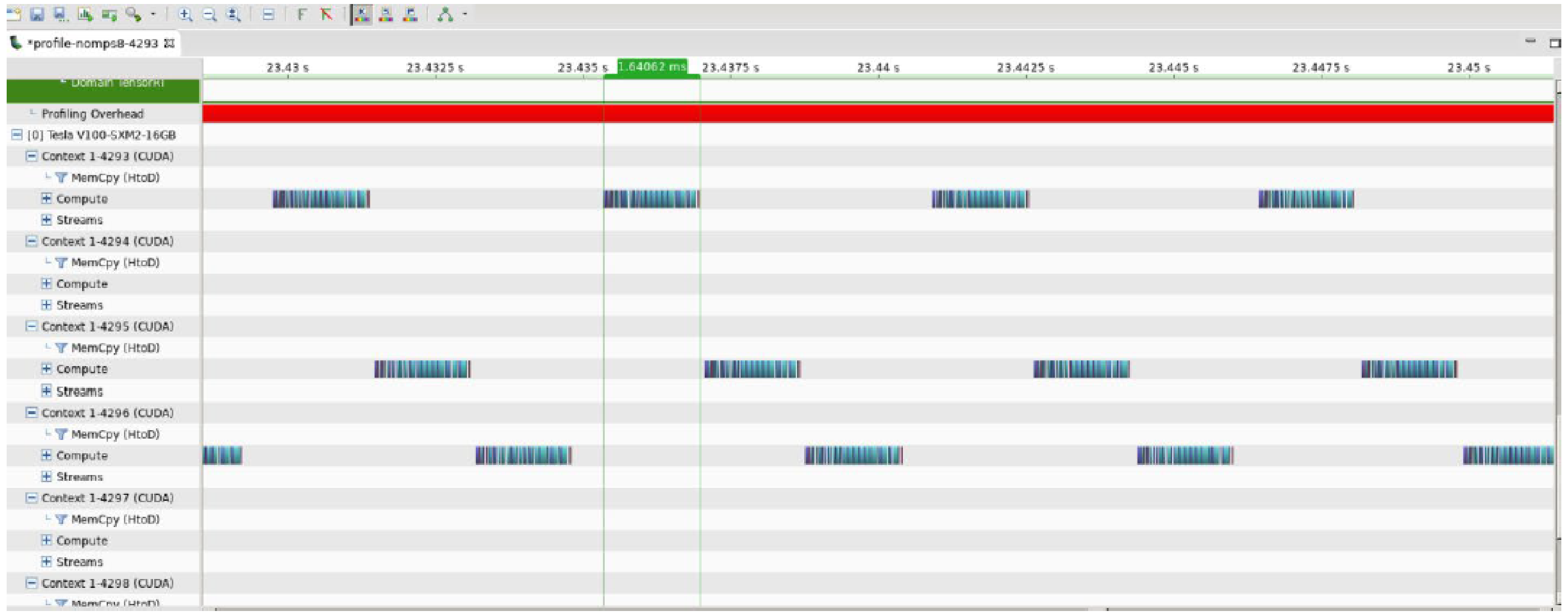
MPS TEST CASE 2

GPU Utilization Metrics – MPS ON



MPS TEST CASE 2

Profiling Analysis



BS=1, NP=8, MPS OFF

BS=1, NP=8, MPS ON

MPS TEST CASE 3

JPEG Resize

JPEG to JPEG resizing is an essential workload for many internet services, including training and inference for image classification, object detection, etc.

And for some service provider, to cut storage expense, they might just storage one image instead of several dozens in different resolutions.

Fastvideo , an NVIDIA Preferred Partner, developed an image processing SDK with CUDA acceleration (one of their customer was Flickr), since there're multi phases in the whole JPEG resize implementation pipeline, like copy from storage to CPU memory, then copy to GPU memory, JPEG decoding, resizing, sharp, JPEG encoding, copy to CPU memory, etc. They've done many optimizations across the whole pipeline, and one technical they adopted is NVIDIA MPS, to optimize the throughput of the GPU system.

We use Fastvideo SDK to perform this testing.

MPS TEST CASE 3

Test Results

Resize JPEG from 1920x1080 to 480x270.

Up to 3.5x throughput improvement when MPS enabled.

Processes Number	FPS – MPS OFF	FPS – MPS ON	Speedup
2	1152	1633	1.42
4	1025	2319	2.26
6	1016	2786	2.74
8	1014	3024	2.98
10	1011	3190	3.15
12	1014	3301	3.25
14	1154	3367	2.92
16	1012	3458	3.42
18	1009	3558	3.53

MPS TEST CASE 3

Test Results

Resize JPEG from 1280x720 to 320x180.

Up to 4.4x throughput improvement when MPS enabled.

Processes Number	FPS – MPS OFF	FPS – MPS ON	Speedup
2	937	2007	2.14
4	904	2910	3.22
6	897	3451	3.85
8	894	3813	4.26
10	890	3848	4.32
12	891	3878	4.35
14	900	3860	4.29
16	889	3921	4.41
18	886	3942	4.45

MULTI-INSTANCE GPU

GPU ARCHITECTURE AND CUDA

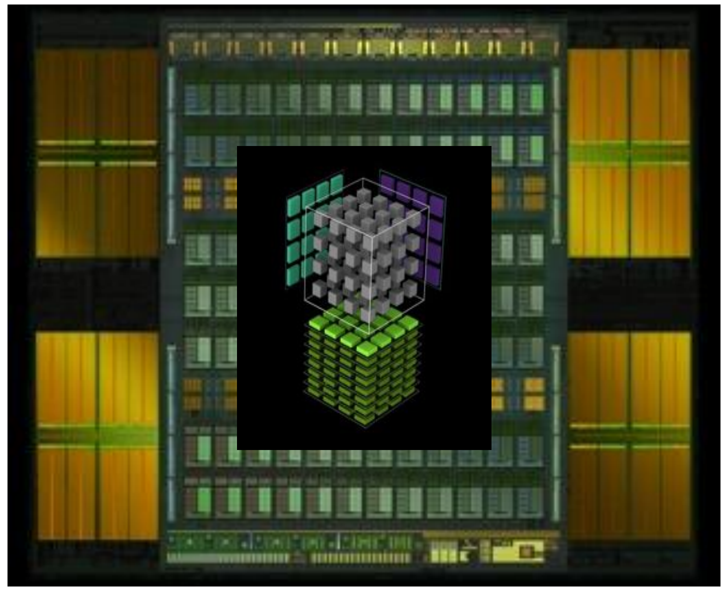
CUDA 8.0

2016

PASCAL

HBM, NVLINK, FP16

CUDA 9.0

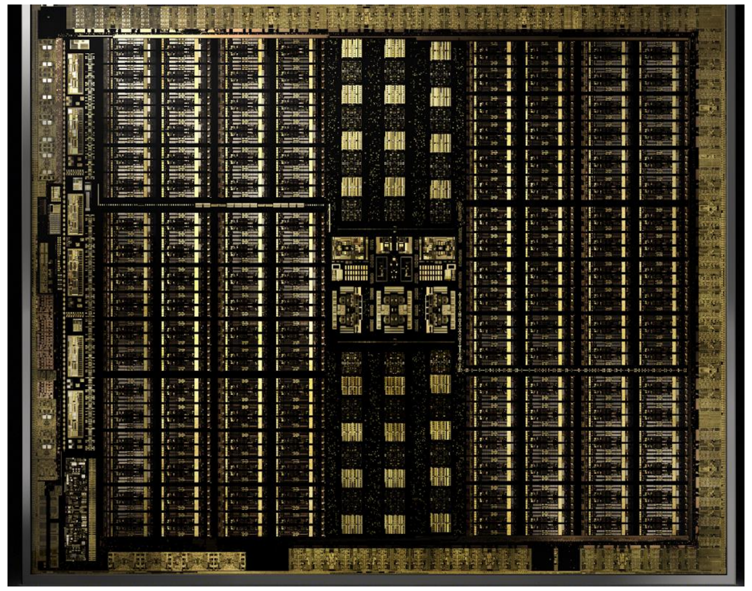


2017

VOLTA

HBM, NVLINK, TENSOR CORES, MPS

CUDA 10.0



2018

TURING

TENSOR CORES, RT CORES

CUDA 11.0

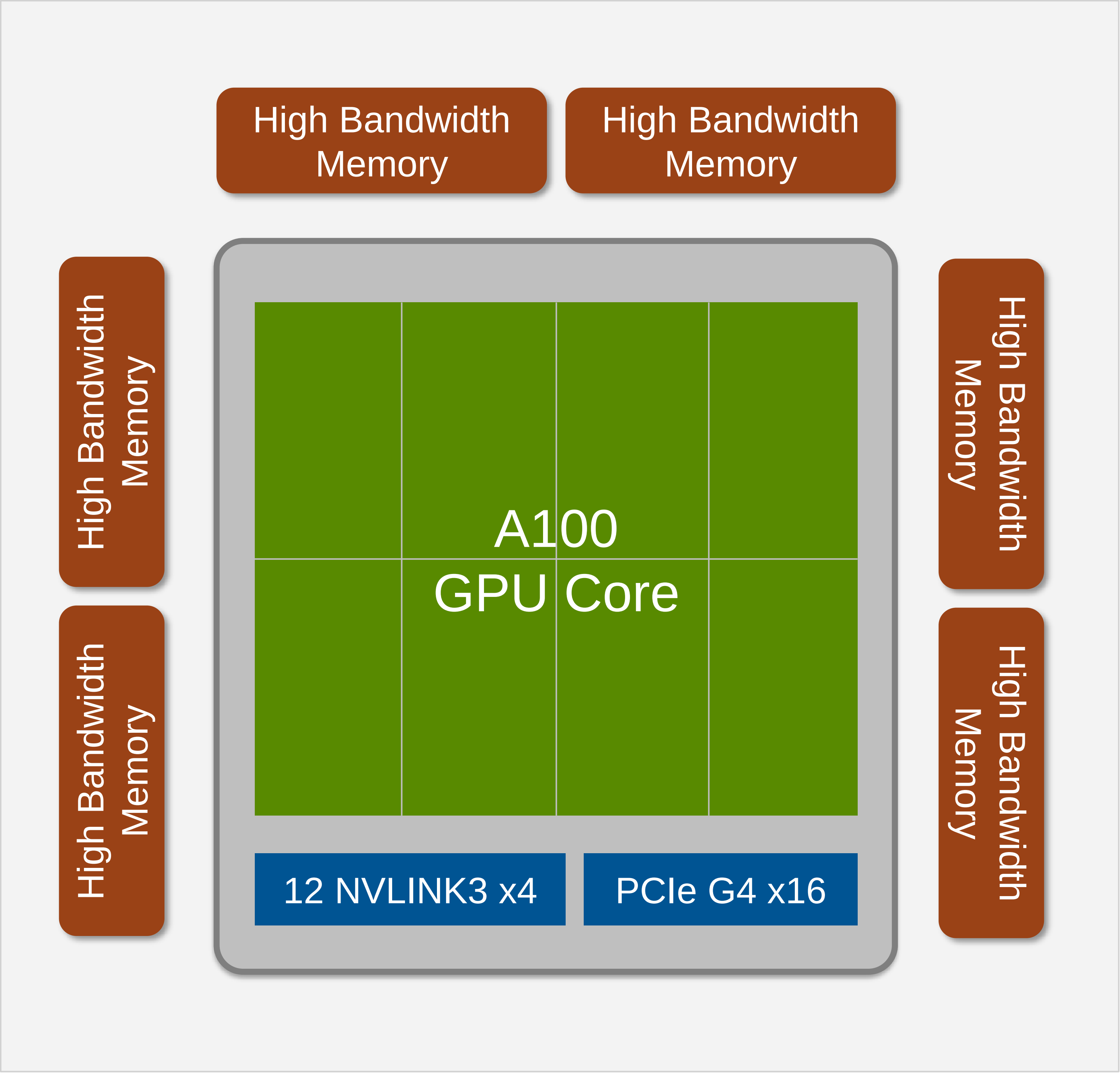
2020

AMPERE

HBM, NVLINK, TENSOR CORES, PARTITIONING

A100 GPU

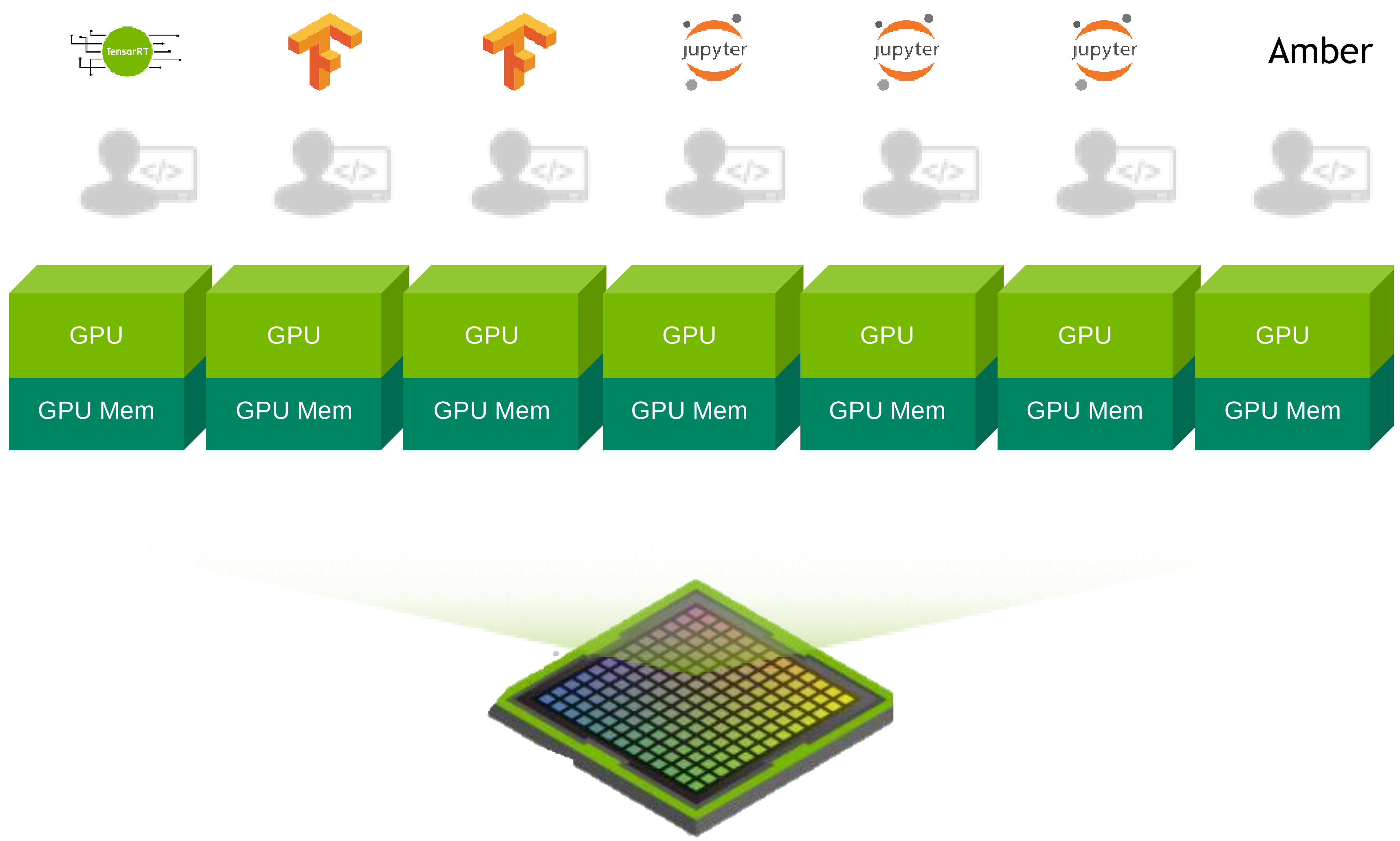
Highest Performance, Efficiency and Utilization



New Technology	Benefit over Volta
Faster Tensor Core for AI, support FP16 & bfloat16	>2x V100 RN50 & Transformer train ~3x Tensor Core FLOPS Dramatically reduce time-to-soln.
New Tensor Core for HPC	2.5x FP64 FLOPS Accelerate core HPC kernels
Wider + Faster Memory	1.7x memory bandwidth Up to 40GB per GPU Larger model & dataset
New NVLINK3 + PCIe Gen4	2x NVLINK bandwidth 2x PCIe bandwidth + SR-IOV
New Multi-Instance GPU, with Fault and Perf Isolation	**Up to 7 concurrent GPUs Higher utilization Substantially lower entry cost**
New Hardware Engines	JPEG HW decoder, 5 video NVDEC Optical flow accelerator

NEW MULTI-INSTANCE GPU (MIG)

Optimize GPU Utilization, Expand Access to More Users with Guaranteed Quality of Service



- Up To 7 GPU Instances In a Single A100: Dedicated SM, Memory, L2 cache, Bandwidth for hardware QoS & isolation
- Simultaneous Workload Execution With Guaranteed Quality Of Service: All MIG instances run in parallel with predictable throughput & latency
- Right Sized GPU Allocation: Different sized MIG instances based on target workloads
- Flexibility: to run any type of workload on a MIG instance
- Diverse Deployment Environments: Supported with Bare metal, Docker, Kubernetes, Virtualized Env.

MIG ISOLATION

Computational Isolation

- SM are not shared between MIGs
- This provides high QoS for each MIG users

DRAM Bandwidth Isolation

- Slices of the L2 cache are physically associated with particular DRAM channels and memory
- Isolating MIGs to non-overlapping sets of L2 cache slices does two things:
  - Isolates BW
  - Allocates DRAM memory between the MIGs

Configuration Isolation

- Creating GPU Instances or Compute Instances do not disturb work running on existing instances

Error Isolation

- Resources within the chip are separately resettable

GPU INSTANCE PROFILES

For A100-SXM4-40GB

GPU Instance	Number of Instances Available	SMs	Memory	NVDECs	Target use-cases	
					Training	Inference
1g.5gb	7	14	5 GB	0	BERT Fine-tuning (e.g. SQuAD), Multiple chatbots, Jupyter notebooks	Multiple inference (e.g. TRITON); ResNet-50, BERT, WnD networks
2g.10gb	3	28	10 GB	1		
3g.20gb	2	42	20 GB	2	Training on ResNet-50, BERT, WnD networks	
4g.20gb	1	56	20 GB	2		
7g.40gb	1	98	40 GB	5		

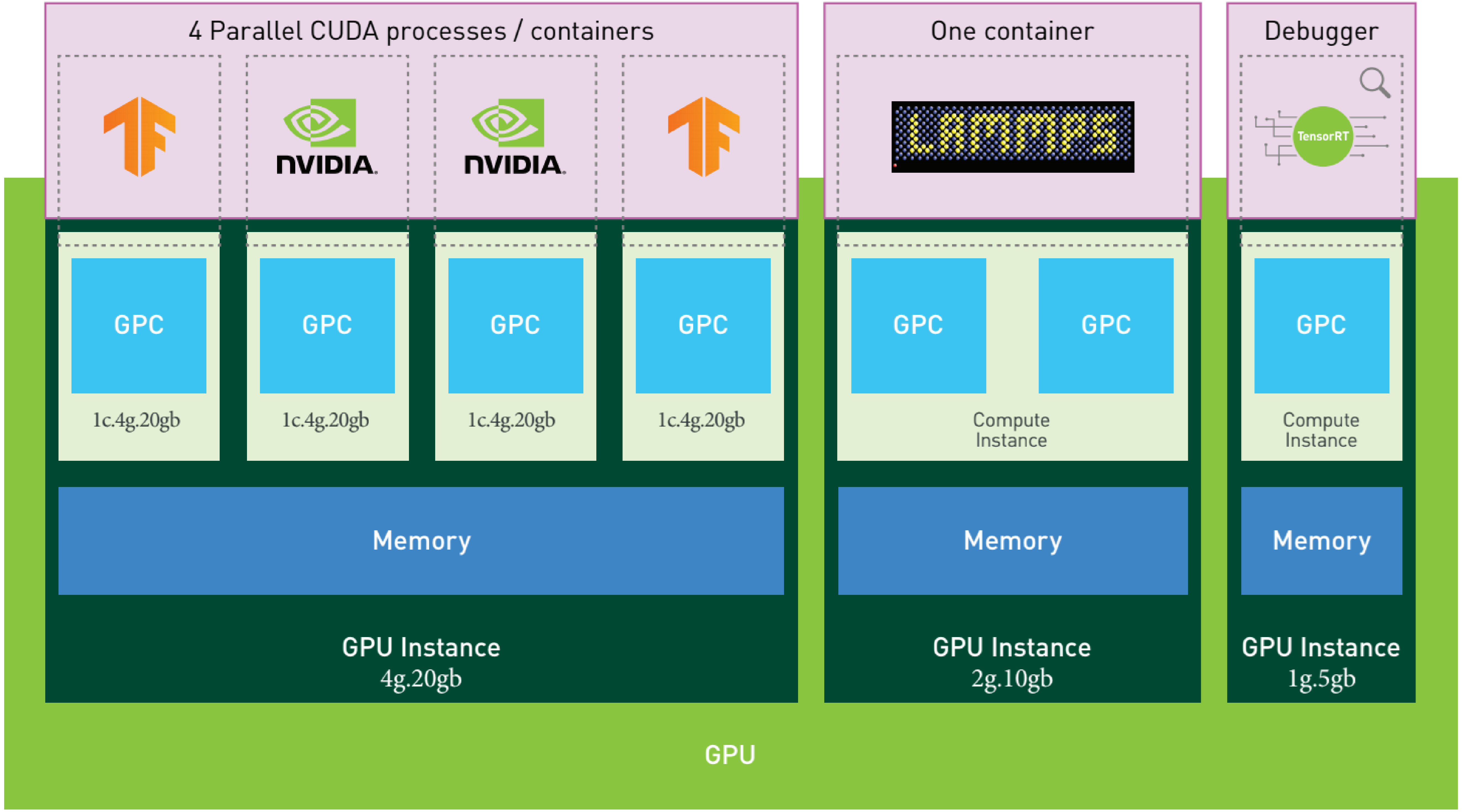
FLEXIBLE MIG CONFIGURATIONS FOR DIFFERENT SCENARIOS

<table>
<tr><th>Slice #1</th><th>Slice #2</th><th>Slice #3</th><th>Slice #4</th><th>Slice #5</th><th>Slice #6</th><th>Slice #7</th></tr>
<tr><td colspan="7">7</td></tr>
<tr><td colspan="4">4</td><td colspan="2">2</td><td>1</td></tr>
<tr><td colspan="4">4</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td colspan="2">2</td><td colspan="2">2</td><td colspan="3">3</td></tr>
<tr><td colspan="2">2</td><td>1</td><td>1</td><td colspan="3">3</td></tr>
<tr><td>1</td><td>1</td><td colspan="2">2</td><td colspan="3">3</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="3">3</td></tr>
<tr><td colspan="3">3</td><td rowspan="3"></td><td colspan="3">3</td></tr>
<tr><td colspan="3">3</td><td colspan="2">2</td><td>1</td></tr>
<tr><td colspan="3">3</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td colspan="2">2</td><td colspan="2">2</td><td colspan="2">2</td><td>1</td></tr>
<tr><td colspan="2">2</td><td colspan="2">2</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td colspan="2">2</td><td colspan="2">2</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td colspan="2">2</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td colspan="2">2</td><td>1</td><td>1</td><td colspan="2">2</td><td>1</td></tr>
<tr><td colspan="2">2</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="2">2</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>

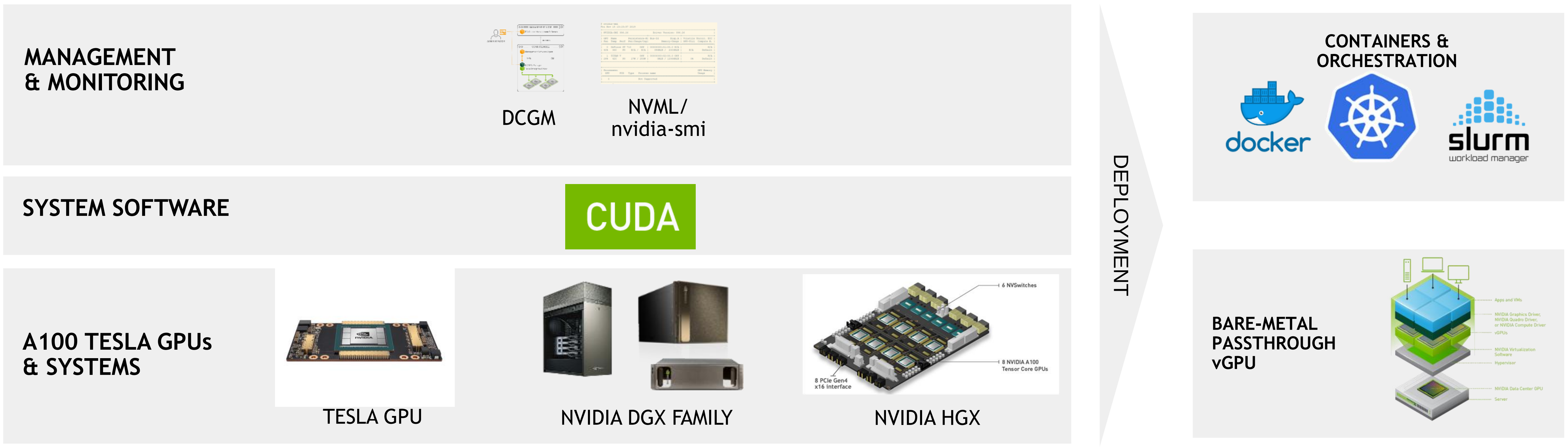
- 18 possible configurations
- NVML or NVIDIA-SMI to create and retire Instance
- Config. can be dynamically updated when the GPU slices involved are idle

EXAMPLE: TWO LEVEL PARTITIONING

GPU Instances and Compute Instances



ENABLEMENT ACROSS SOFTWARE STACK



- Support for bare-metal and containerized environments
  - Interaction directly via NVML/nvidia-smi
  - Kubernetes (device enumeration, resource type), Slurm
  - Docker CLI
- Monitoring and management (including device metrics association to MIG)

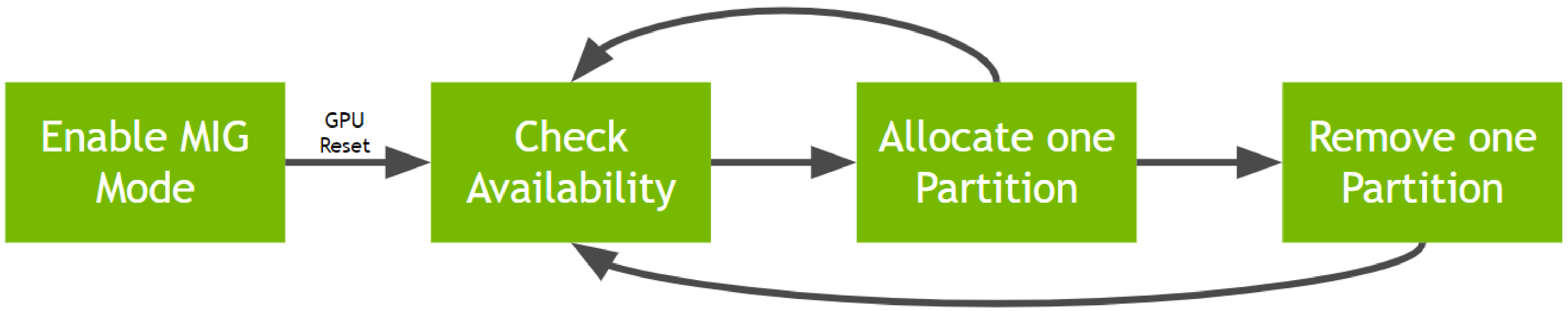
USER WORKFLOW: MIG MANAGEMENT

List/Create/Update/Destroy Instances via NVML and nvidia-smi

GPU reset required to enable/disable MIG mode (one-time operation)

Use NVML/nvidia-smi (even through containers) to manage MIG

Example: Create new instance with nvidia-smi




```
# nvidia-smi mig --list-gpu-instances
+--------------------------------------------------+
| GPU instances:                                   |
| GPU    Name        Profile  Instance   Placement |
|                      ID       ID       Start:Size |
|==================================================|
|   0  1g.5gb        19        9          2:1     |
+--------------------------------------------------+
|   0  1g.5gb        19       10          3:1     |
+--------------------------------------------------+
|   0  1g.5gb        19       13          6:1     |
+--------------------------------------------------+
|   0  2g.10gb       14        3          0:2     |
+--------------------------------------------------+
|   0  2g.10gb       14        5          4:2     |
+--------------------------------------------------+
```

MIG: RUNNING DOCKER CONTAINERS

User Workflow

- Run GPU containers with MIG using "--gpus" option in Docker 19.03
  - Primarily for single node development and testing
- Enabled via NVIDIA Container Toolkit (previously known as nvidia-docker2)
- Users configure MIG partitions using NVML/nvidia-smi
- Launching the container requires specifying the GPU instances to expose to the container

```
$ docker run \
  --gpus '"device=0:0,0:1"' \
  nvidia/cuda:11.0-base nvidia-smi -L

GPU 0: A100-SXM4-40GB (UUID: GPU-2ceff3df-31b3-caf2-eace-a494b4b7926b)
  MIG 3g.20gb Device 0: (UUID: MIG-GPU-2ceff3df-31b3-caf2-eace-
a494b4b7926b/1/0)
  MIG 3g.20gb Device 1: (UUID: MIG-GPU-2ceff3df-31b3-caf2-eace-
a494b4b7926b/2/0)

$ docker run \
  --gpus '"device=MIG-GPU-2ceff3df-31b3-caf2-eace-a494b4b7926b/1/0"' \
  nvidia/cuda:11.0-base nvidia-smi -L

GPU 0: A100-SXM4-40GB (UUID: GPU-2ceff3df-31b3-caf2-eace-a494b4b7926b)
  MIG 3g.20gb Device 0: (UUID: MIG-GPU-2ceff3df-31b3-caf2-eace-
a494b4b7926b/1/0)
```

MIG: RUNNING CONTAINERS USING K8S

User Workflow

- MIG configured on the node ahead of time
- Expected to be transparent to the end user
- Simple exposure model for homogenous nodes
- Other exposure options still in discussion and not settled yet
- User jobs will be able to only execute on a single Compute Instance

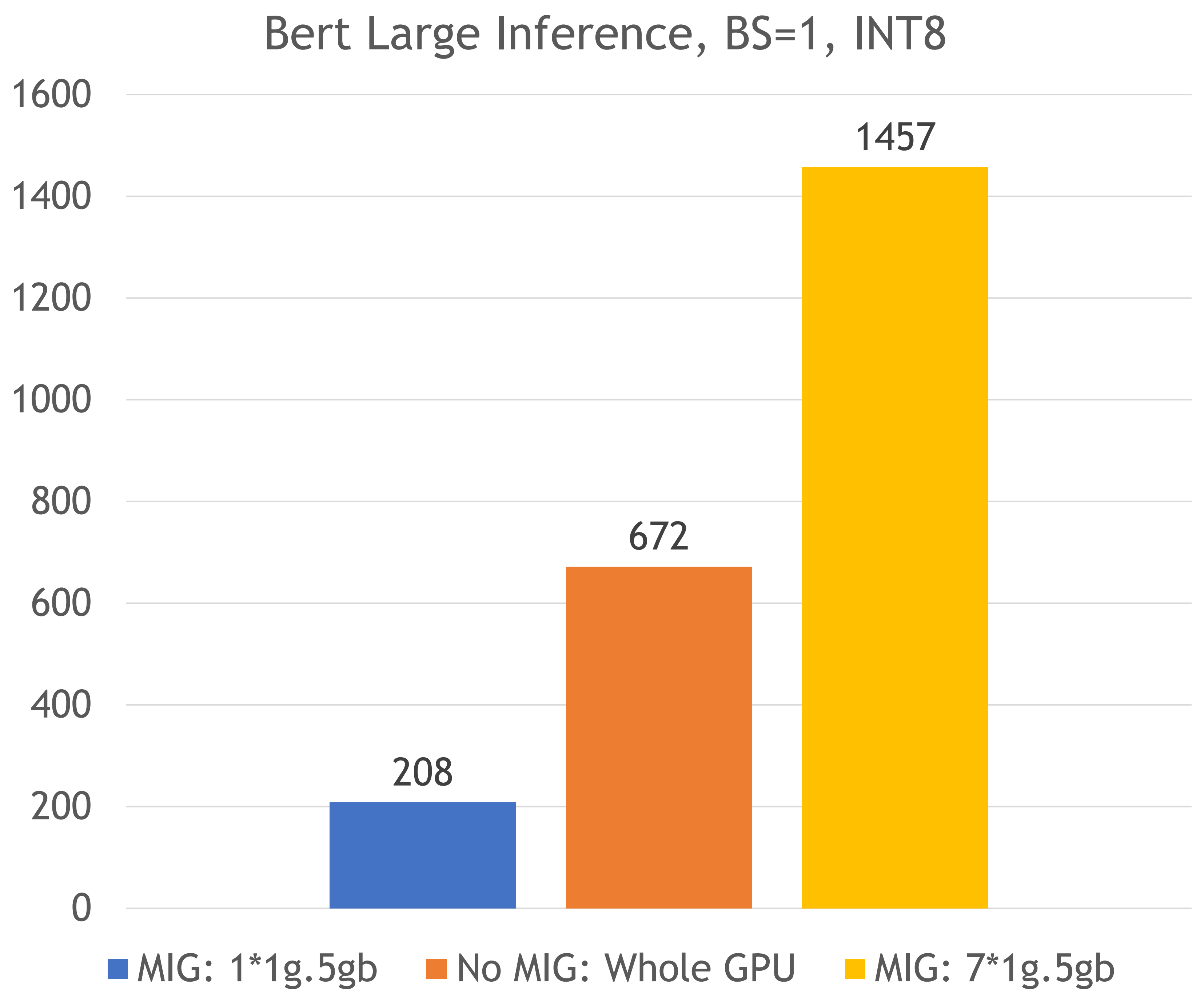
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda:11.0-base
      resources:
        limits:
          nvidia.com/gpu: 1
  nodeSelector:
    nvidia.com/gpu.product: A100-SXM4-40GB-MIG-1g.5gb
    nvidia.com/cuda.runtime: 11.0
    nvidia.com/cuda.driver: 450.28.0
```

MIG TEST CASE 1 – BERT LARGE INFERENCE

Test Results

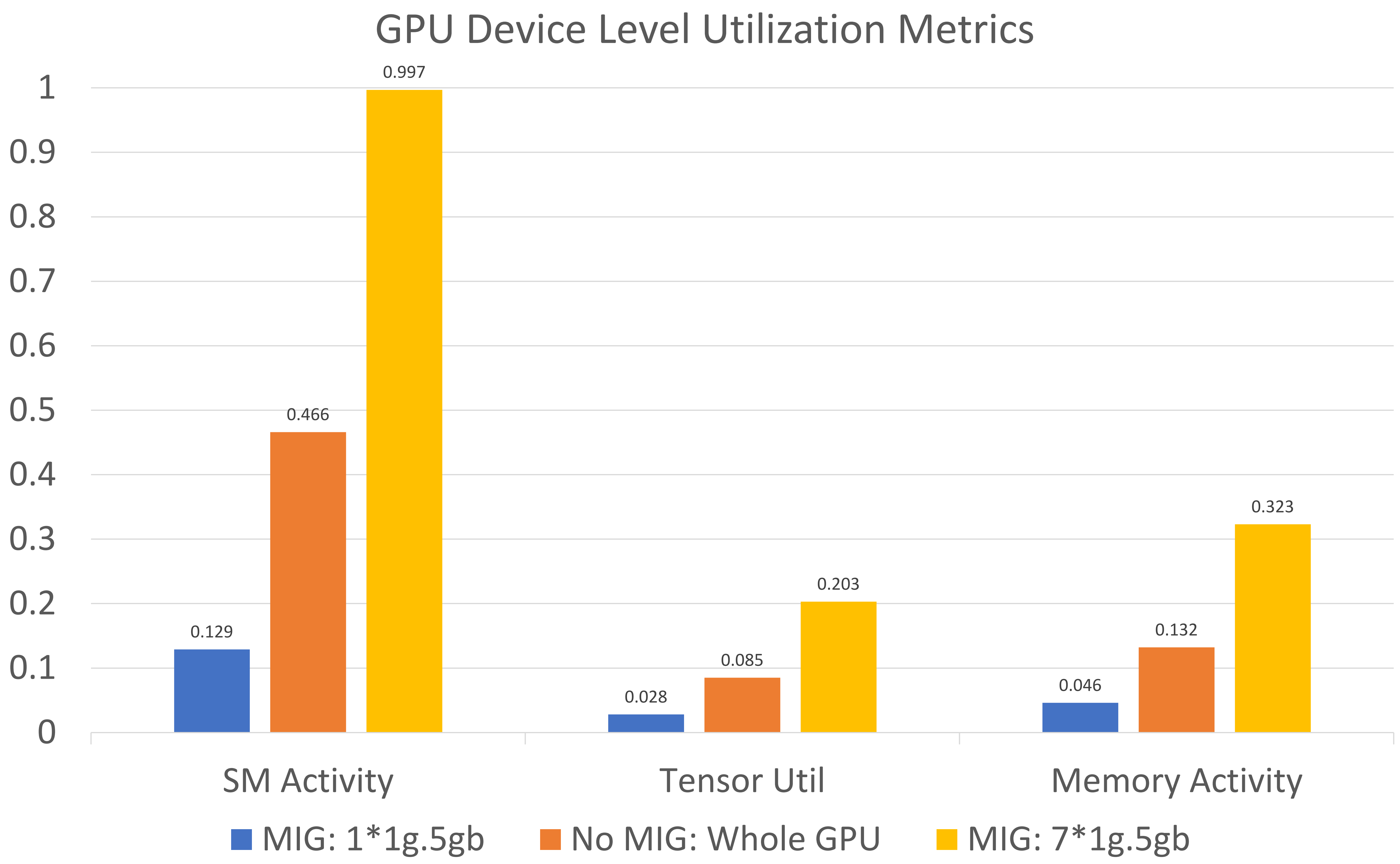
Perf among 7 MIG 1g.5gb slice is very stable and consistent. MIG provides great perf isolation and QoS.

2.1x throughput when MIG is enabled for this case and config.



MIG TEST CASE 1 - BERT LARGE INFERENCE

GPU Utilization Metrics



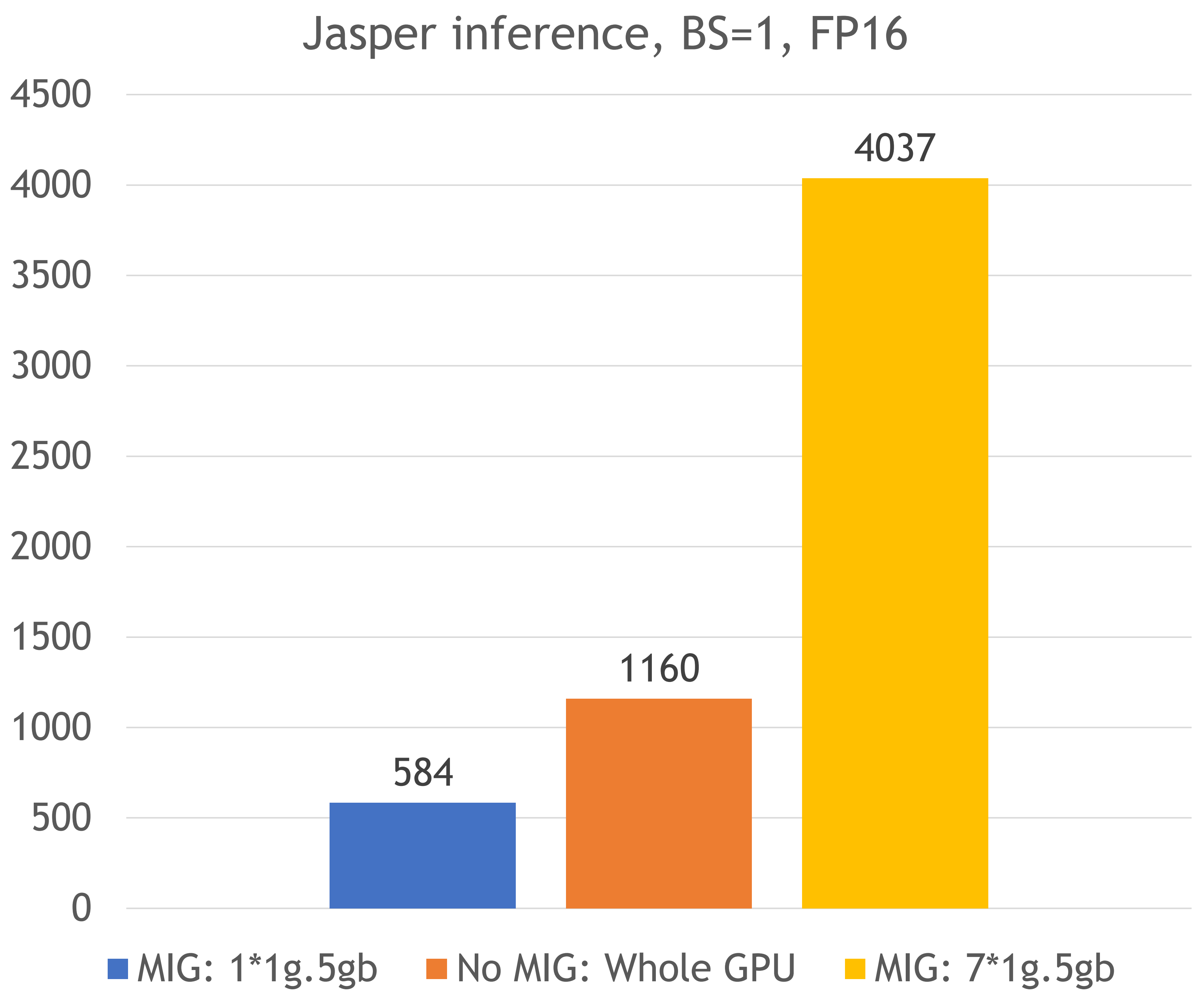
NVIDIA

MIG TEST CASE 2 – JASPER INFERENCE

Test Results

Throughput: amount of audio seconds processed by GPU in one second

With MIG enabled, throughput up to 3.4x improvement.

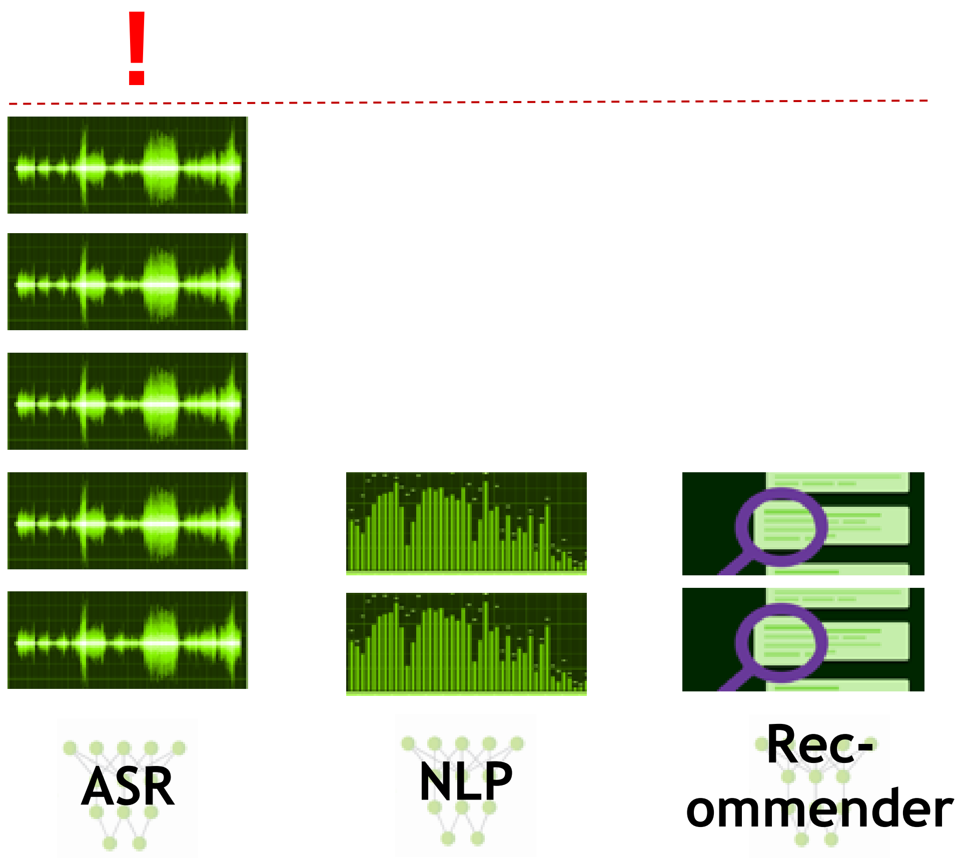


TRITON AND VGPU BRIEF

INEFFICIENCY LIMITS INNOVATION

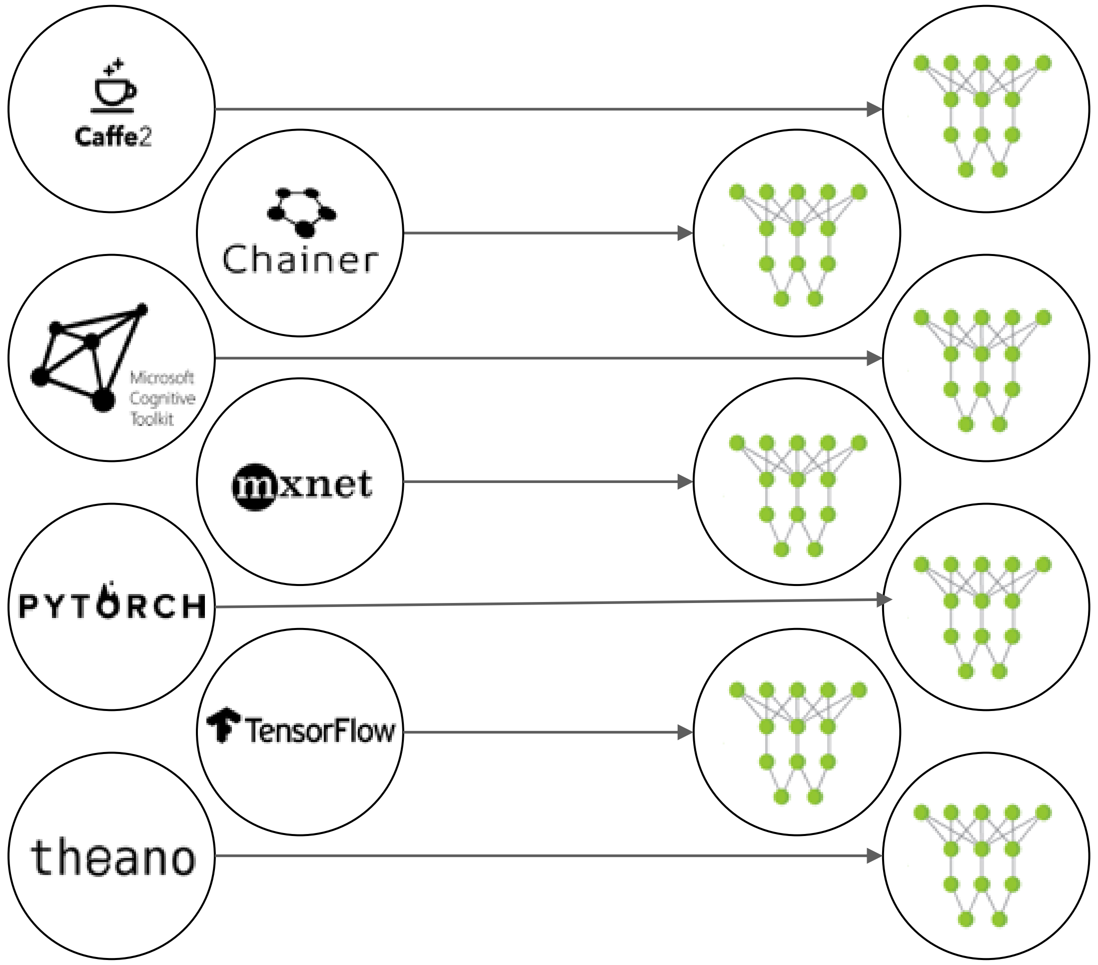
Difficulties with Deploying Data Center Inference

Single Model Only



Some systems are overused while
others are underutilized

Single Framework Only



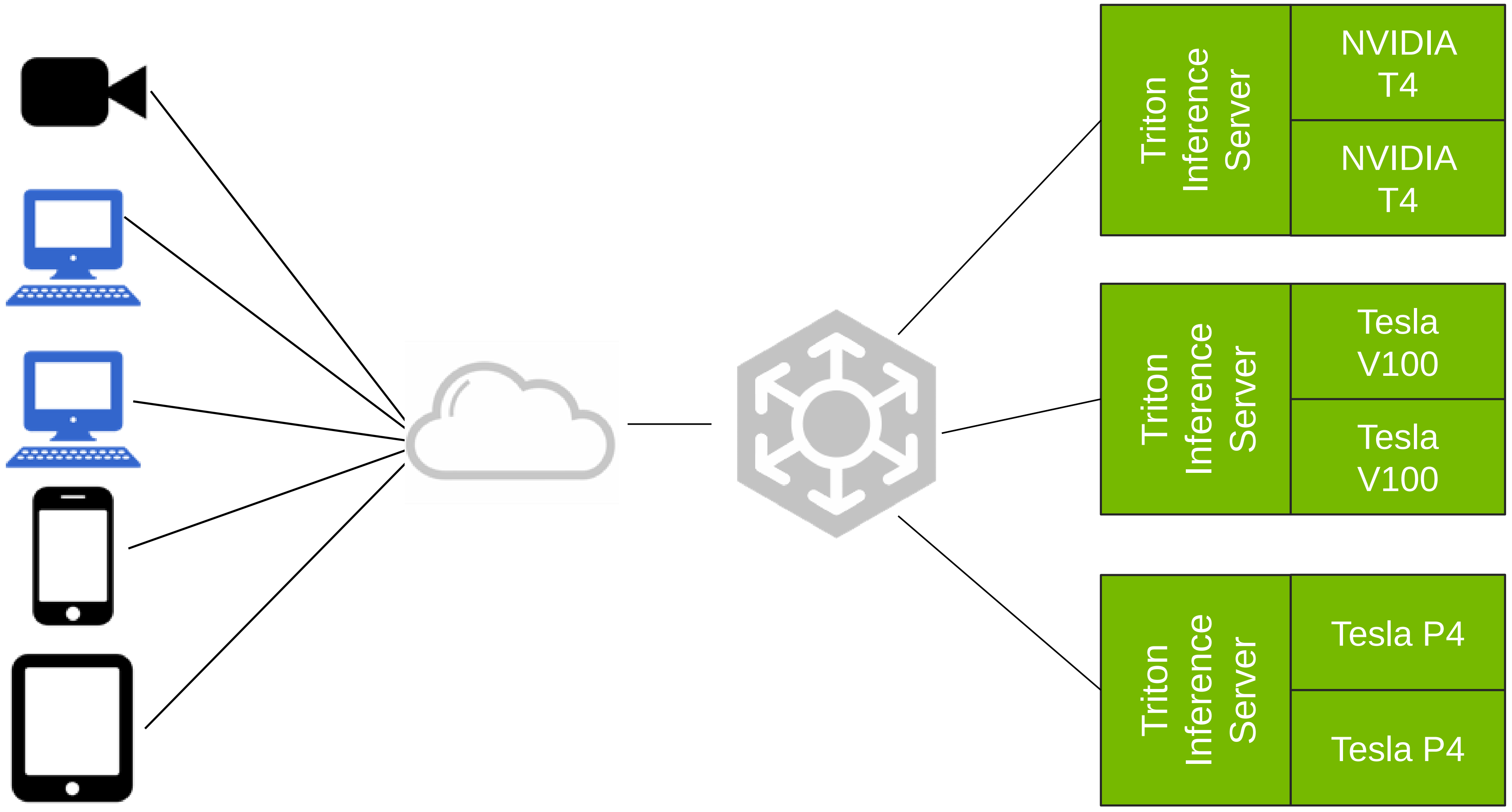
Solutions can only support
models from one framework

Custom Development

Developers need to reinvent the
plumbing for every application

NVIDIA TRITON INFERENCE SERVER

Production Data Center Inference Server



Maximize real-time inference performance of GPUs

Quickly deploy and manage multiple models per GPU per node

Easily scale to heterogeneous GPUs and multi GPU nodes

Integrates with orchestration systems and auto scalers via latency and health metrics

Now open source for thorough customization and integration

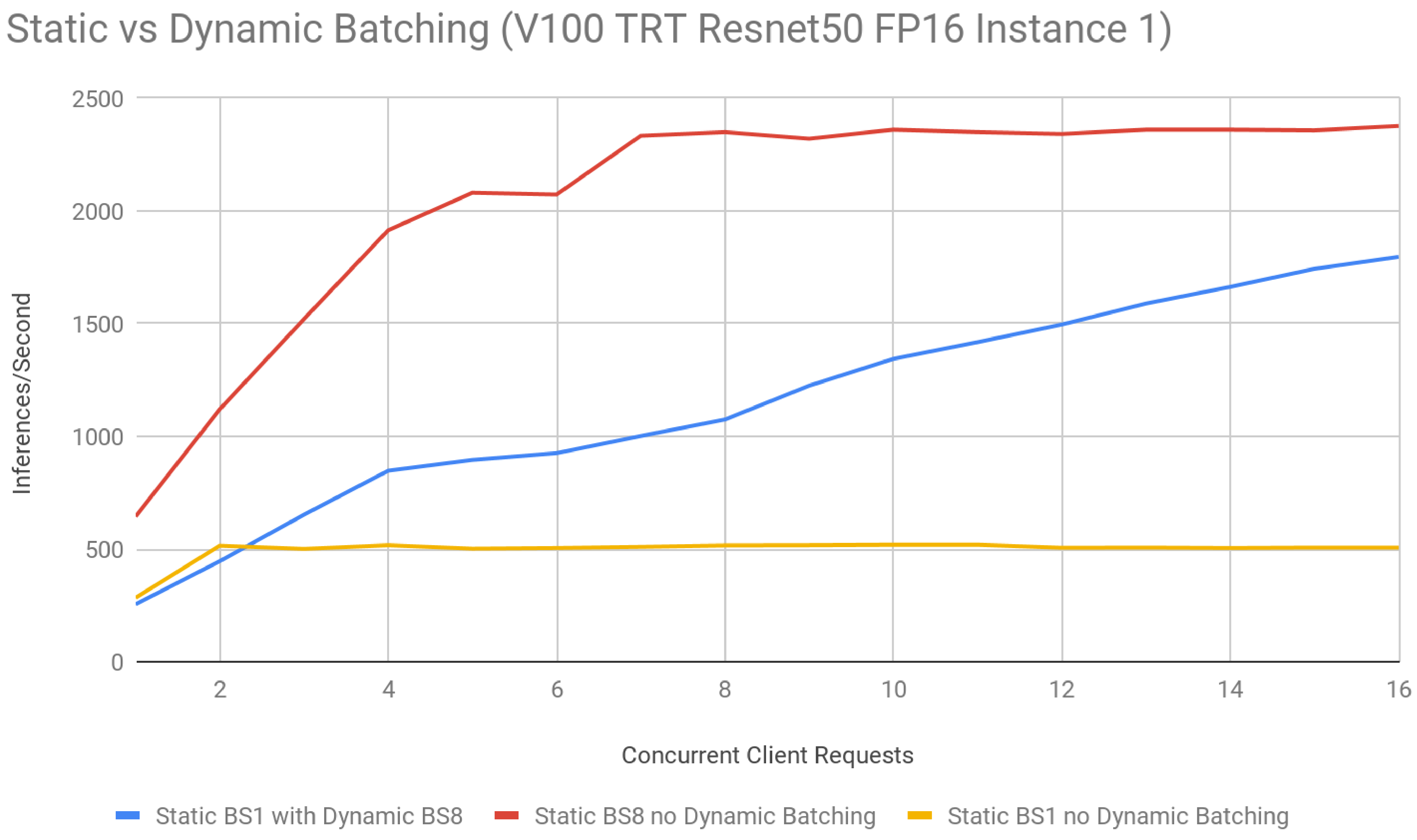
DYNAMIC BATCHING

2.5X Faster Inferences/Second at a 50ms End-to-End Server Latency Threshold

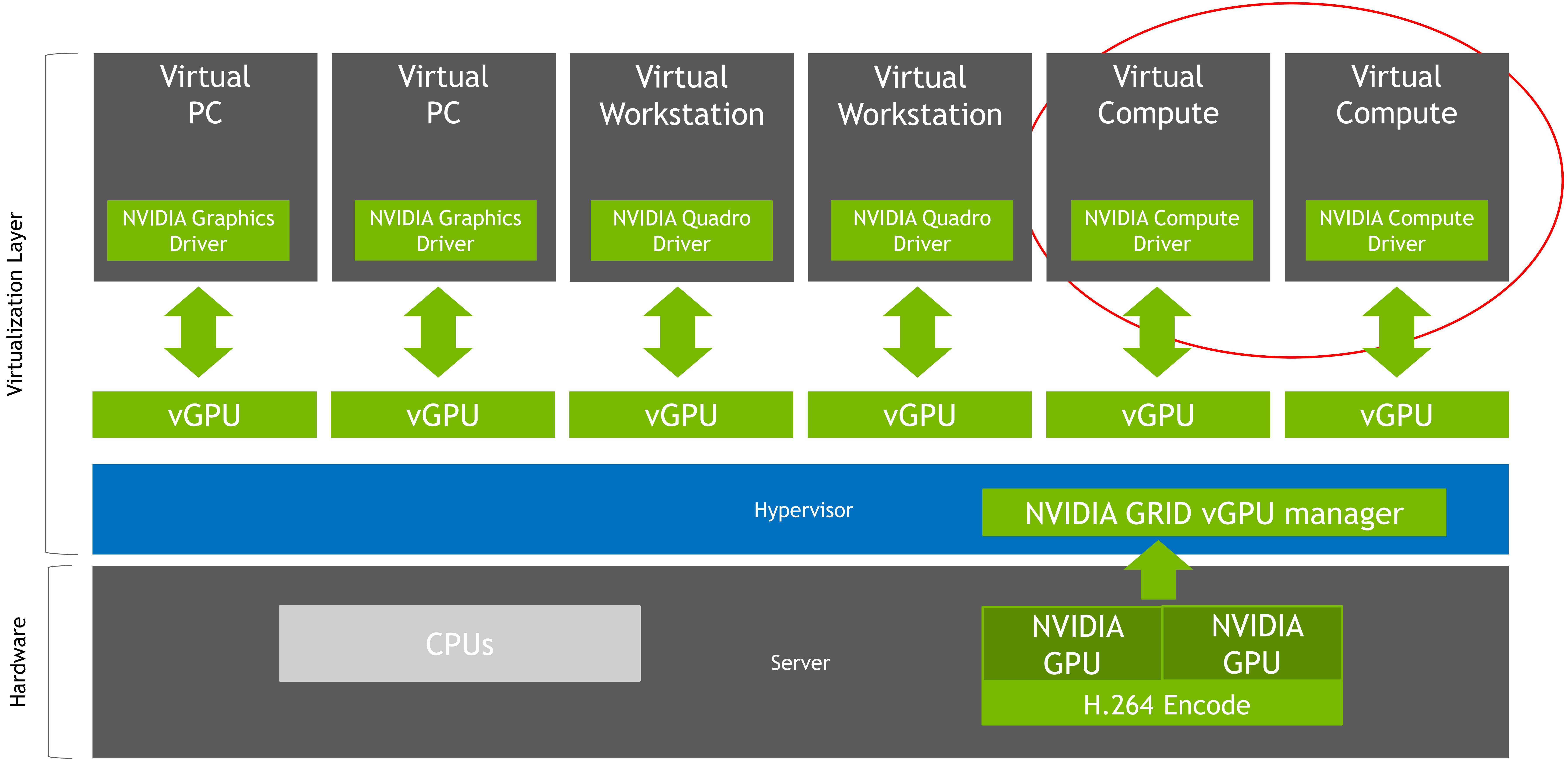
Triton Inference Server groups inference requests based on customer defined metrics for optimal performance

Customer defines
1) batch size (required)
2) latency requirements (optional)

Example: No dynamic batching (batch size 1 & 8) vs dynamic batching



VGPU FOR GRAPHICS AND COMPUTING



VGPU FOR COMPUTING

vCS

Hypervisor provides best security, isolation guarantee.

vCS provides a good option for cost sensitive customers and those new comers to GPU computing, or application of low-utilized GPU scenarios.

Flexible scheduler strategy: Best effort, fixed-share, equal-share.

Flexible scheduler time slice (1-20 ms controllable).

Perf is guaranteed even that it’s time-round sharing for SM resources.

QUICK SUMMARY

CUDA CONCURRENCY MECHANISMS

Triton, MPS, vGPU and MIG

	Parallel work	Address space isolation	SM performance isolation	Memory performance isolation	Error isolation
TRITON (CUDA Streams)	Yes	No	No	No	No
MPS	Yes	Yes	Yes (by percentage, not partitioning)	No	No
vGPU	Yes	Yes (With hypervisor)	Yes (Time-slicing)	Yes	Yes
MIG	Yes	Yes	Yes	Yes	Yes

COMPARISON
Part 1

Simple Comparison Among MPS, vGPU, TRITON, MIG				
	MPS	vGPU	TRITON	MIG
Intro Link	MPS Whitepaper	Official Link	Github	MIG Whitepaper-NDA
Open Source	No	No	Yes	No
Free	Yes	No	Yes	Yes
Main Positioning	Improve GPU utilization for applications that doesn't fully utilize GPU, by schedule multi-process, with limited execution resource.	Offer a consistent user experience for every virtual workflow and improve GPU utilization in some scenario, by split GPU into multiple vGPUs as memory size equal partition, by integrating with hypervisor (virtual machine technology).	Provide a cloud inferencing solution optimized for NV GPU, with an inference service via HTTP or gRPC endpoint.	Improve GPU utilization and serve more users with physical resource isolation and QoS guarantee.
Target Applications	Applications that doesn't fully utilize GPU: HPC-MPI application, training, inference with small matrix size.	3D Rendering, vGaming, training, inference.	Inference.	Training, inference, HPC.

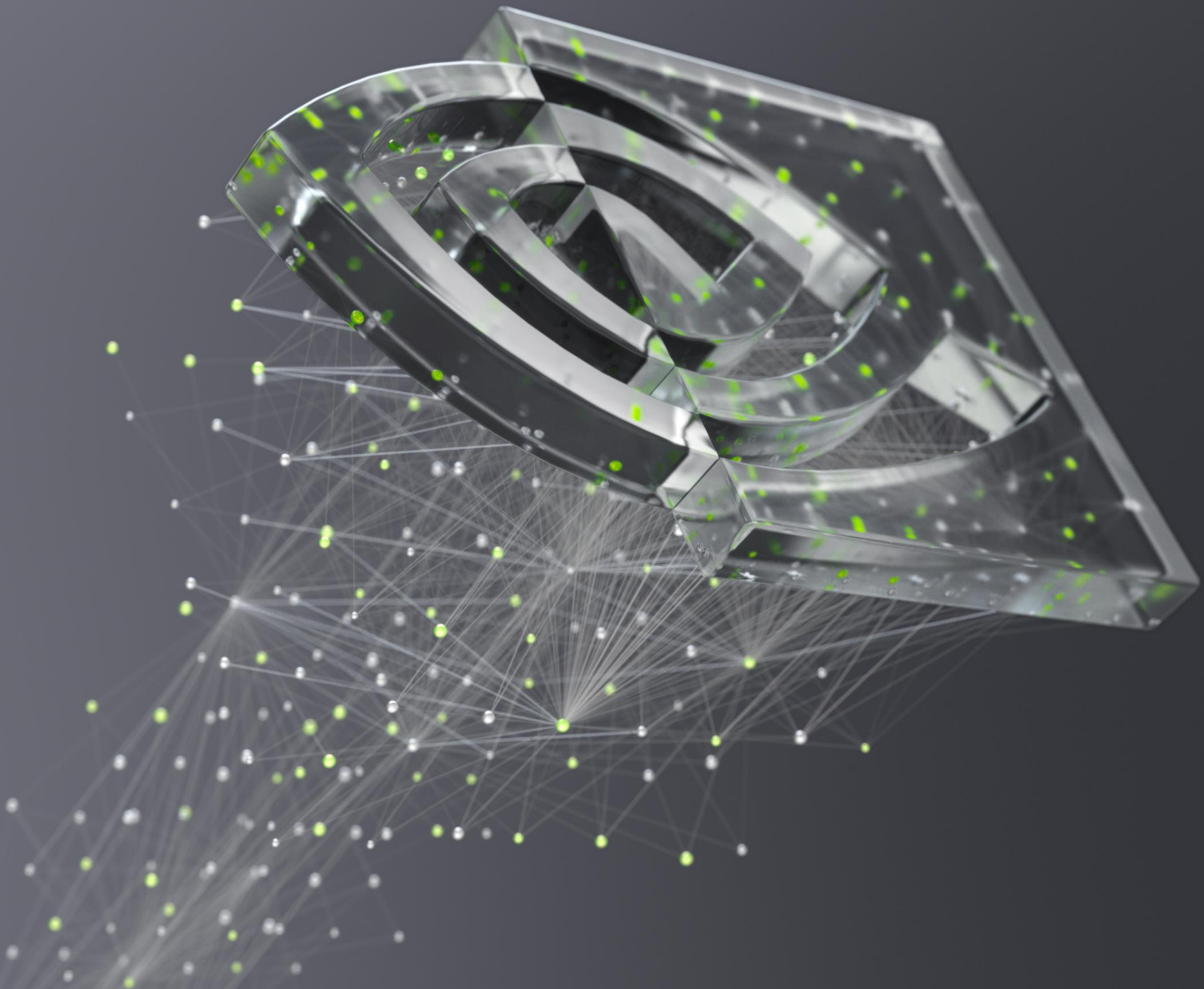
COMPARISON

Part 2

Simple Comparison Among MPS, vGPU, TRITON, MIG				
	MPS	vGPU	TRITON	MIG
Supported GPU	GPU since Kepler	P100, P40, P4, P6, V100, T4, RTX8000, RTX6000, M10, M60	All GPU	A100
Supported OS	Linux	Linux, Windows	Linux	Linux
Extra Software Needed	No	Hypervisor(KVM, Citrix, VMWare, etc)	No	No
Benefits	Improve GPU utilization, improve throughout	Improve GPU utilization via time-sharing, improve user experience	Improve GPU utilization, improve throughout	Improve GPU utilization, improve throughput, serve more users, provide QoS and fault isolation.
GPU Resource Isolation	Context level isolation, memory and SM sharing	GPU memory isolation, SM sharing by rotation.	TRTIS executes model(app) instance as Thread(CPU)-Stream(GPU). SM sharing is via multi-stream.	GPU memory isolation, SM isolation, other engines isolation(CEs, NVDEC).

COMPARISON
Part 3

Simple Comparison Among MPS, vGPU, TRITON, MIG				
	MPS	vGPU	TRITON	MIG
QoS	No strong guarantee	Guarantee in time-slicing sharing envelop	No strong guarantee	Strong, the best guarantee
Ease of Use	Easy	Medium	Easy	Easy
Support	Forum	Professional team	Github issue	Professional team
Considerations/Limitations	No fault tolerance. Really not suitable for arbitrary combination of multi-user applications, especially for public cloud scenario with full isolation requirements.	Not really sharing SM as this is a time-sharing/slicing implementation.	Mainly confined to inference type workloads. Multi-streaming currently not effective to TF based models (limiting factor from TensorFlow).	Only for compute workloads in MIG mode, don't support P2P between GPU compute instances.
Correlations	MPS, vGPU, TRITON, MIG are not mutually exclusive solutions. Example: you can run MPS or TRITON in vGPU environment. Example: you can run MPS or vGPU in MIG-enabled A100 system. Example: you can even run multi processes in TRITON with MPS enabled, under vGPU with MIG-enabled system.			



NVIDIA