



KubeCon



CloudNativeCon

Europe 2023





KubeCon



CloudNativeCon

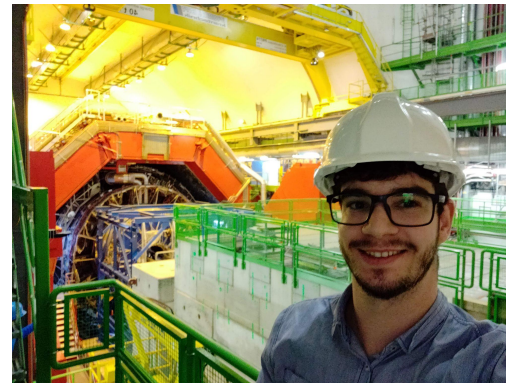
Europe 2023

Efficient Access to Shared GPU Resources: Mechanisms and Use Cases

Diogo Guerra & Diana Gaponcic, CERN



Diogo Guerra - diogo.filipe.tomas.guerra@cern.ch
Computing Engineer with the Kubernetes team @ CERN



Diana Gaponcic - diana.gaponcic@cern.ch
Computing Engineer with the Kubernetes team @ CERN





CMS

CERN

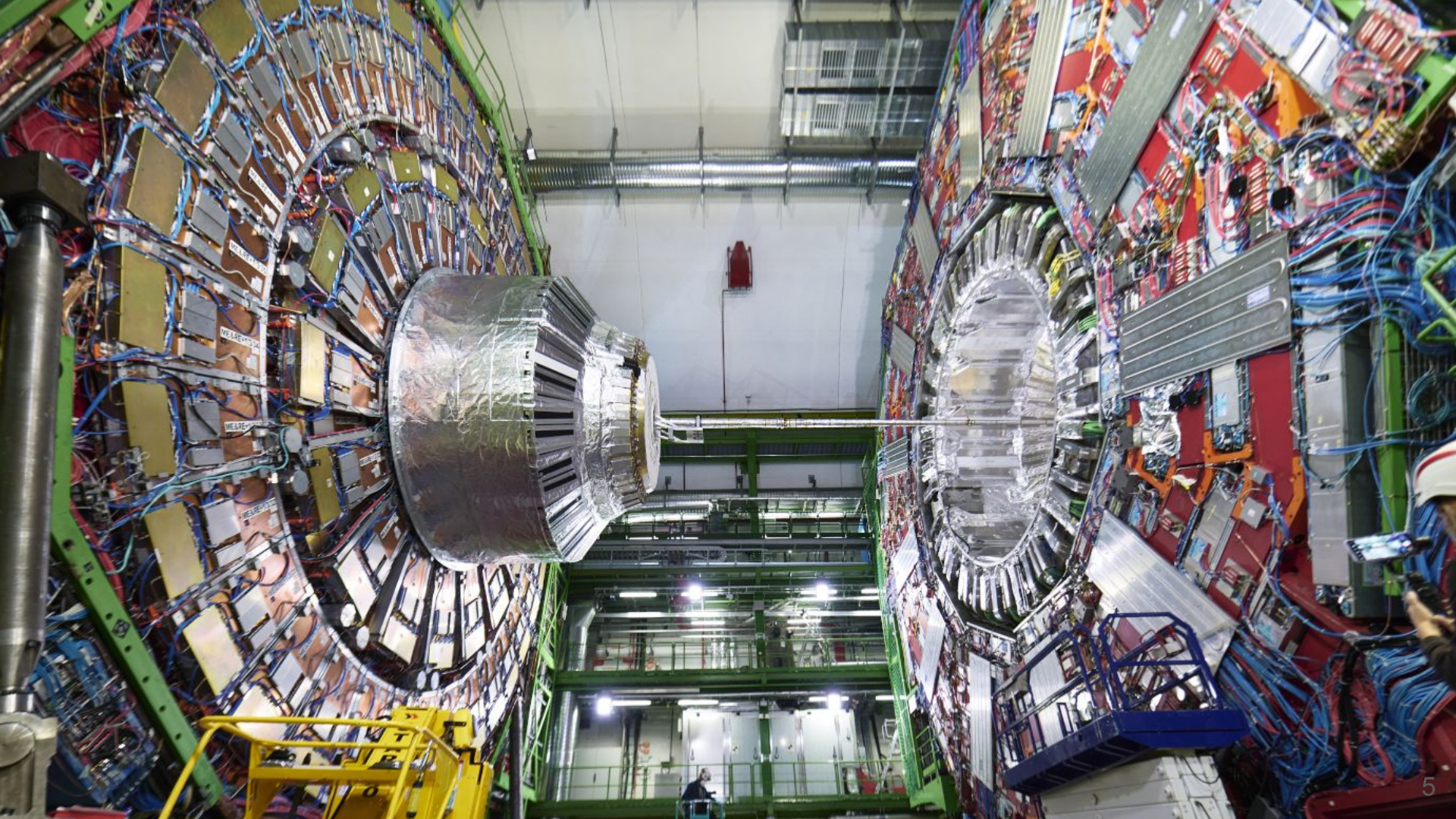
LHC

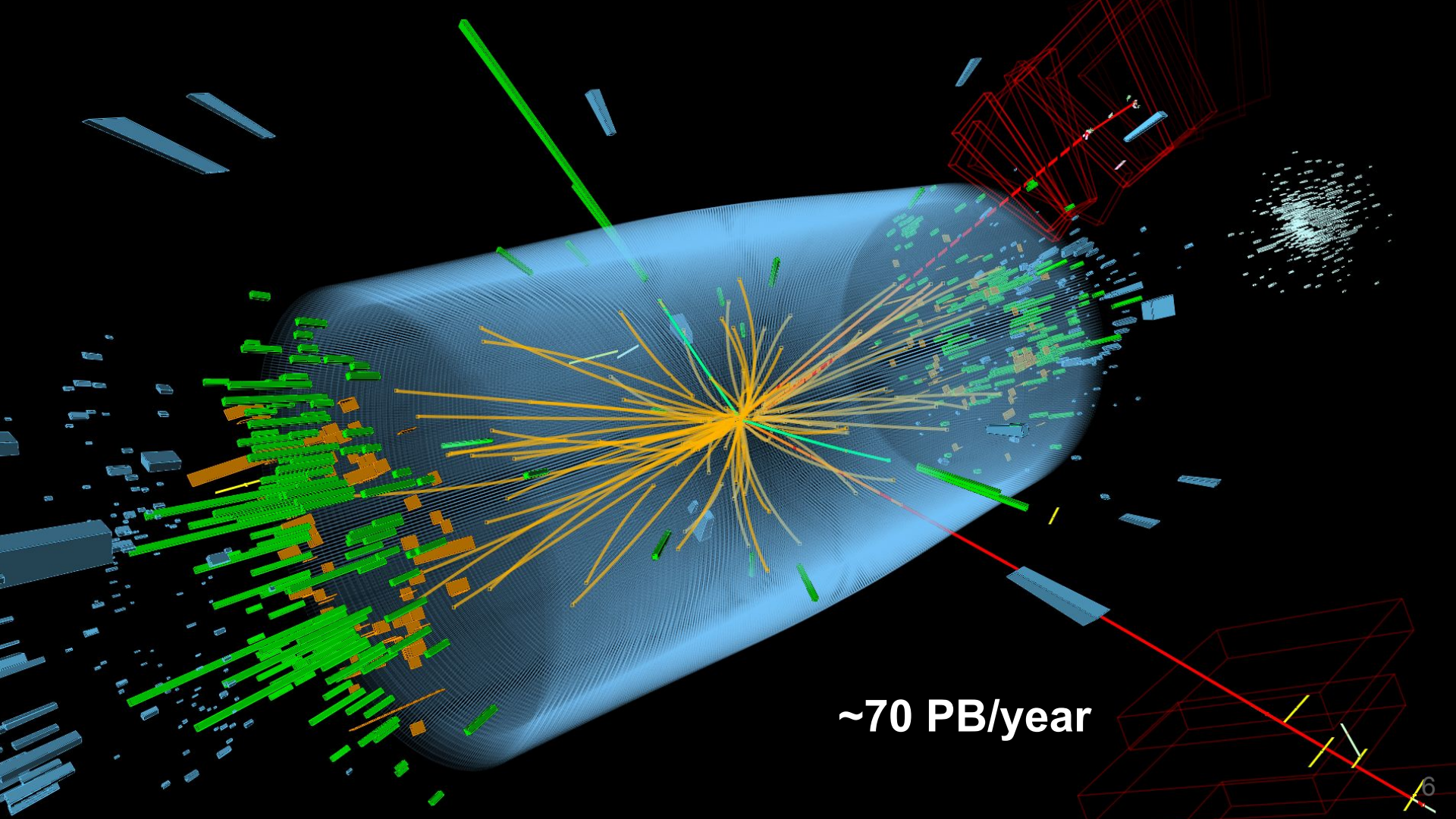
[Large Hadron Collider]

ALICE

LHCb

ATLAS





$\sim 70 \text{ PB/year}$

Some of our use cases for GPUs:

- Simulation
- Event filtering
- Machine Learning:
 - a. [event reconstruction \(GNN\)](#)
 - b. [faster generation of simulation data \(3DGAN\)](#)
 - c. [reinforcement learning for beam calibration](#)



CERN Alice GPU Trigger



CERN LHCb High-Level Trigger

Efficient GPU usage challenges:

- Suboptimal code
- Existing workloads' usage patterns not designed with GPUs in mind
- Spiky workloads (ex: Development workflow using interactive tools like Notebooks)

Infrastructure perspective:

- GPU power density constraints
- Vendor lock-in (ex: cuda, oneapi from Intel, frameworks built on cuda)
- Limited resources to meet users' demand



KubeCon



CloudNativeCon

Europe 2023

Story time!

Meet the actors

Mr Brown

Badly coded simulation job:

1. Low average GPU usage (CPU dependant workload)
2. Needs 10 GiB VRAM (8 + 2 dynamic)
3. Long running process



Mr Pink

An inference service which is occasionally triggered by outside events:

- Spiky and unpredictable execution
- Mostly sits idle
- Saturates the GPU cores
- Max 10 GiB VRAM (2 + 8 dynamic)



Mr Orange

Never know what to expected from the notebook users:

1. Potential memory leaks
2. Poorly considered batch size
3. GPU memory locked by an idle notebook





KubeCon



CloudNativeCon

Europe 2023

Use Case 1:

User is assigned full GPU card



KubeCon



CloudNativeCon

Europe 2023

Let's onboard Mr Brown



Let's onboard Mr Brown

- GPU underutilized
- Steady memory utilization ~ 20%

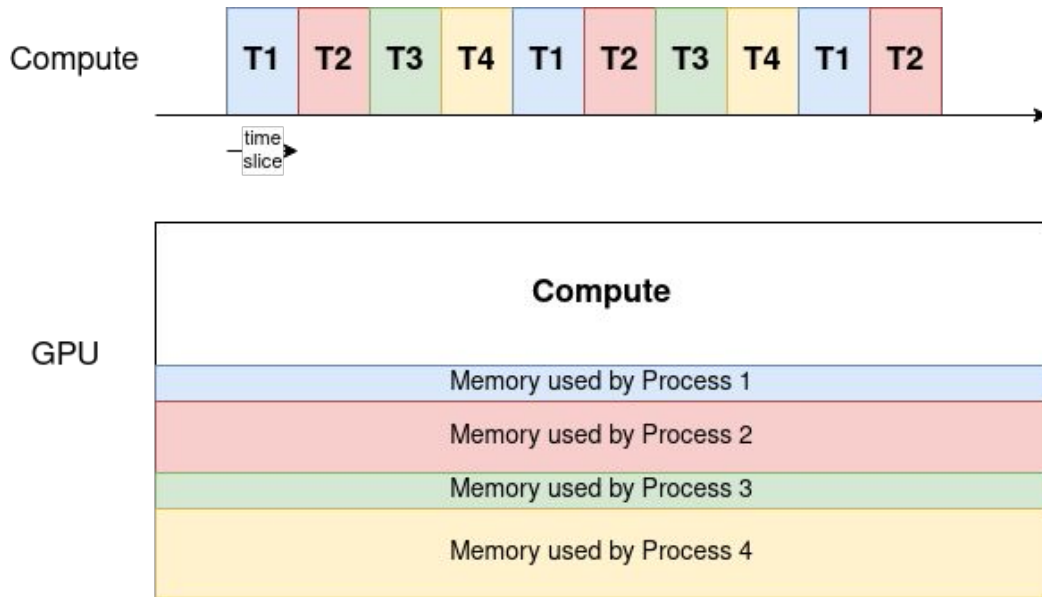


Use Case 2:

Sharing GPUs with Time Slicing

What is Time Slicing

- The scheduler gives an equal share of time to all GPU processes and alternates them in a round-robin fashion.
- The memory is shared between the processes
- The compute resources are assigned to one process at a time



Time Slicing config

```
# values.yaml in NVIDIA gpu operator Helm chart
```

```
...  
devicePlugin:  
  config:  
    name: nvidia-time-slicing-config
```

```
# $ cat nvidia-time-slicing-config.yaml
```

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: nvidia-time-slicing-config  
  namespace: kube-system  
data:  
  slice-4: |-  
    version: v1  
    sharing:  
      timeSlicing:  
        renameByDefault: true  
        failRequestsGreaterThanOne: true  
        resources:  
        - name: nvidia.com/gpu  
          replicas: 4
```

When set to true, each resource is advertised under the name <resource-name>.shared instead of <resource-name>.

Allocatable:

```
...  
nvidia.com/gpu: 1
```

Allocatable:

```
...  
nvidia.com/gpu: 0  
nvidia.com/gpu.shared: 4
```



KubeCon



CloudNativeCon

Europe 2023

Now let's onboard Mr Pink



Let's onboard Mr Brown and Mr Pink

- improved GPU utilization
- better memory consumption (~ 50 %)





KubeCon



CloudNativeCon

Europe 2023

Let's risk...

and onboard Mr Orange



The GPU is fully utilized

- GPU utilization 100%
- memory consumption increasing





Mr Brown
cause of death: **OOM**



Mr Pink
cause of death: **OOM**

Time Slicing conclusions

Advantages

Works on a wide range of NVIDIA architectures

An easy way to set up GPU concurrency

An unlimited number of partitions

Disadvantages

No process or memory isolation

No ability to set priorities

Inappropriate for latency-sensitive applications (ex: desktop rendering for CAD workloads)



KubeCon



CloudNativeCon

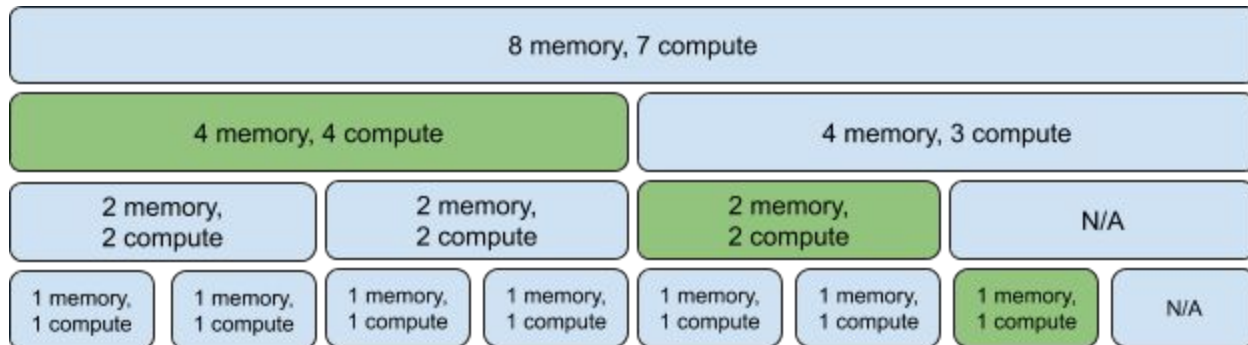
Europe 2023

Can we do better?

Yes we can!

Introducing MIG

Multi Instance GPU (MIG) can partition the GPU into up to seven instances, each **fully isolated** with its own high-bandwidth memory, cache, and compute cores.



NVIDIA A100 GPU can be partitioned into up to 7 MIG devices

source: http://www.pattersonconsultingtn.com/blog/introduction_to_nvidia_mig.html

```
# values.yaml in NVIDIA gpu operator Helm chart
```

```
...
```

```
mig:
```

```
  strategy: mixed
```

```
  migManager:
```

```
    config:
```

```
      name: nvidia-mig-config
```

mixed strategy enumerates resource types for every MIG device available.

```
apiVersion: v1
```

```
kind: ConfigMap
```

```
metadata:
```

```
  name: nvidia-mig-config
```

```
data:
```

```
  config.yaml: |
```

```
    version: v1
```

```
    mig-configs:
```

```
      # A100-40GB
```

```
      3g.20gb-2x2g.10gb:
```

```
        - devices: all
```

```
          mig-enabled: true
```

```
          mig-devices:
```

```
            "2g.10gb": 2
```

```
            "3g.20gb": 1
```

Allocatable:

...

nvidia.com/gpu: 1

Allocatable:

...

nvidia.com/gpu: 0

nvidia.com/mig-2g.10gb: 2

nvidia.com/mig-3g.20gb: 1



instance type:
2g.10gb

2/7 Cores
2/8 Memory

instance type:
2g.10gb

2/7 Cores
2/8 Memory



instance type:
3g.20gb

3/7 Cores
4/8 Memory

GPU utilization with MIG

Every process:

- is isolated
- saturates own resources
- cannot influence other processes



Now Mr Orange is isolated

It would have
been nice to
live this long in
the movie too!



Alive!
Hooray!



Advantages

Hardware isolation allows processes to run securely in parallel and not influence each other

Monitoring and telemetry data available at partition level

Allows partitioning based on use cases, making the solution flexible

Disadvantages

Only available for Ampere and Hopper architecture (as of mid 2023)

Reconfiguring the partition layout requires all running processes to be evicted

Potential loss of available memory depending on chosen profile layout



KubeCon



CloudNativeCon

Europe 2023

Performance Tradeoffs?

Let's benchmark

Benchmarking Setup

Environment:

- NVIDIA A100 40GB PCIe GPU
- Kubernetes version 1.22
- Cuda version utilized: 11.6
- Driver Version: 470.129.06



Xsuite



Benchmarked Script

Benchmarked script:

- Simulation script that generates collision events
- Built with Xsuite (Suite of python packages for multiparticle simulations for particle accelerators)
- Very heavy on GPU usage
- Low on memory accesses
- Low on CPU-GPU communication

Time Slicing performance penalty

Number of particles	Passthrough [seconds]	Shared x1 [seconds]	Loss [%]
10 000 000	51.135	51.93	1.98
15 000 000	76.374	77.12	0.97
20 000 000	99.55	99.91	0.36
30 000 000	151.57	152.61	0.68

Conclusion:

The performance loss when enabling time slicing (shared x1) is negligible.

Time Slicing performance penalty

Number of particles	Shared x1 [seconds]	Expected Shared x2 = Shared x1 * 2 [seconds]	Actual Shared x2 [seconds]	Loss [%]
10 000 000	51.93	103.86	138.76	33.6
15 000 000	77.12	154.24	212.71	37.90
20 000 000	99.91	199.82	276.23	38.23
30 000 000	152.61	305.22	423.08	38.61

Conclusion:

If the GPU needs to perform context switching (going from shared x1 to shared x2) it leads to a **performance loss of ~38%**.

Time Slicing performance penalty

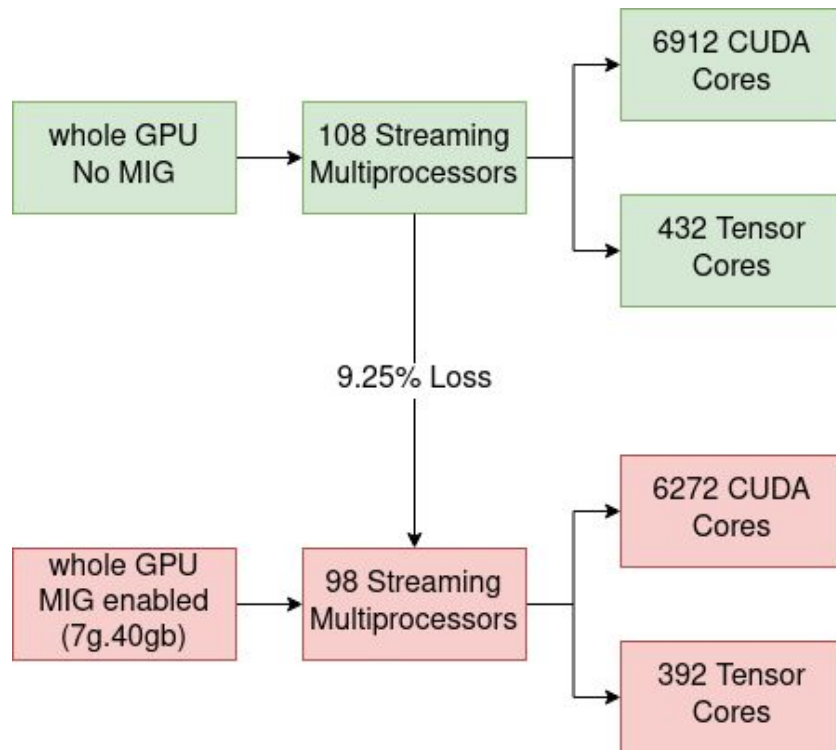
Number of particles	Shared x2 [seconds]	Shared x4 [seconds]	Loss [%]
10 000 000	138.76	281.98	1.6
15 000 000	212.71	421.55	0
20 000 000	276.23	546.19	0
30 000 000	423.08	838.55	0

Number of particles	Shared x4 [seconds]	Shared x8 [seconds]	Loss [%]
10 000 000	281.98	561.98	0
15 000 000	421.55	838.22	0
20 000 000	546.19	1087.99	0
30 000 000	838.55	1672.95	0

Conclusion:

Further increasing the number of processes (shared x4, shared x8), doesn't introduce additional performance loss.

MIG performance penalty for A100



MIG performance penalty

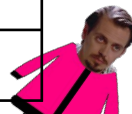
Number of particles	Whole GPU, no MIG [seconds]	Whole GPU, with MIG (7g.40gb) [seconds]	Loss [%]
5 000 000	26.365	28.732	8.97 %
10 000 000	51.135	55.930	9.37 %
15 000 000	76.374	83.184	8.91 %

Conclusion:

The theoretical loss of 9.25% can be seen experimentally as well.

MIG performance penalty

Number of particles	7g.40gb [s]	3g.20gb [s]	2g.10gb [s]	1g.5gb [s]
5 000 000	28.732	62.268	92.394	182.32
10 000 000	55.930	122.864	183.01	362.10
15 000 000	83.184	183.688	273.7	542.3











Number of particles	3g.20gb / 7g.40gb	2g.10gb / 3g.20gb	1g.5gb / 2g.10gb
5 000 000	2.16	1.48	1.97
10 000 000	2.19	1.48	1.97
15 000 000	2.20	1.48	1.98
ideal scale	$7/3 = 2.33$	$3/2 = 1.5$	$2/1 = 2$

Conclusion:

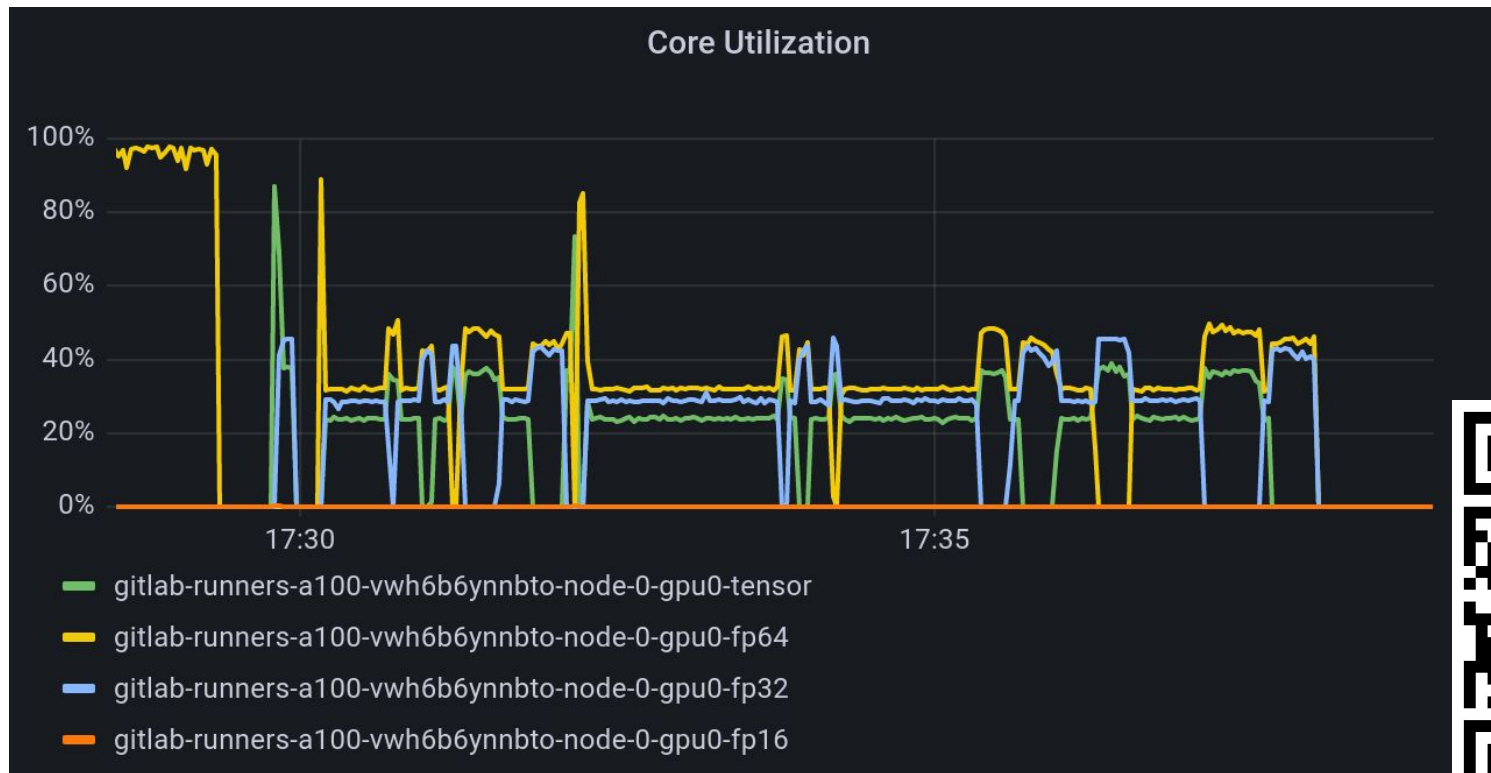
The scaling between partitions converges to the ideal values.

Final conclusions

Category	Examples	Time slicing	MIG
Latency sensitive	CAD, Engineering Applications		
Interactive	Notebooks	 ¹	
Performance intensive	Simulation		
Low priority	CI Runners		

¹ Independent workloads can trigger OOM errors between each other. Needs an external mechanism to control memory usage (similar to kubelet CPU memory checks)

Nice to know/have - Profiling Metrics



Grafana Dashboard

Profiling the A100 compute pipeline utilization

References:



KubeCon



CloudNativeCon

Europe 2023



CERN k8s GPU blog posts



NVIDIA Documentation

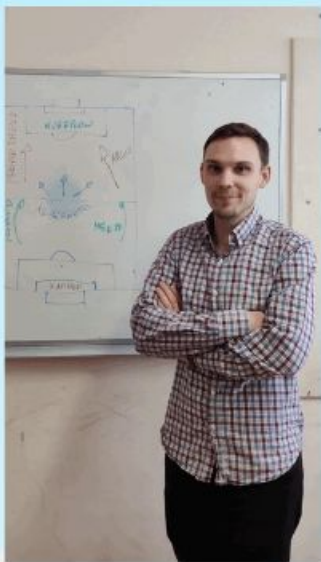


k8s NVIDIA gpu-operator

Special thanks:



Ricardo Rocha



Dejan Golubovic



KubeCon



CloudNativeCon

Europe 2023

Acknowledgements:

- Reservoir Dogs (1992)





KubeCon



CloudNativeCon

Europe 2023

Thank you!