nodeflux
Extending Vision Beyond Imagination

GPU on Kubernetes

Eufrat Tsaqib

www.nodeflux.io

INTRO

Eufrat Tsaqib

- Software Engineer at Nodeflux
- In-charge of Nodeflux Snapshot (Cloud)
- Just graduated from Physics major
- Latest research on Hyperspectral Imaging

Nodeflux

- Perusahaan berbasis kecerdasan buatan (AI) untuk melakukan analisis video dan gambar.
- Perusahaan pertama Indonesia di bidang Intelligent Video Analytics (IVA).
- Official NVIDIA Global Partner.

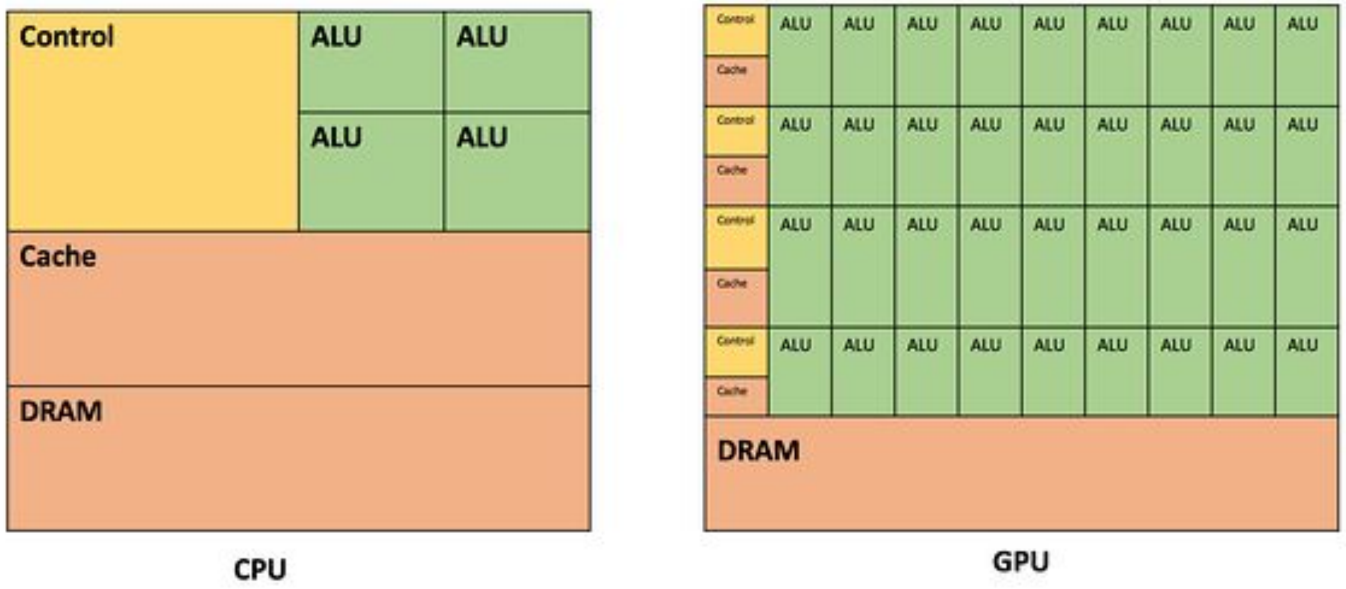
CONTENTS

WHAT IS A GPU

- Used to accelerate image or graphics computation
- Highly parallel mechanism
- Good to compute large matrix/tensor
- As an additional extension to computer
- Scarce resource, especially one designed for ML (eg. NVIDIA Tesla, AMD Instinct)

WHAT IS A GPU

Highly parallel mechanism

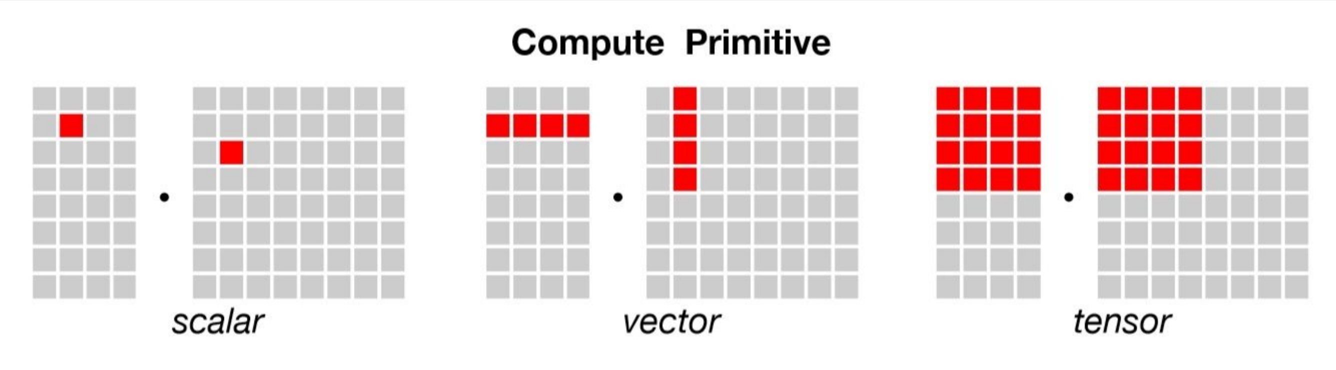


A GPU has more Arithmetic Logic Units (ALU) than a typical CPU.

- Increased ability to process simple operations in parallel

WHAT IS A GPU

Good to compute large matrix/tensor



CPU: 1 X 1 data unit
GPU: 1 X N data unit
TPU: N X N data unit

```
with tf.device('/gpu:0'):
  a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
  b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
c = tf.matmul(a, b)
```

WHAT IS A GPU

Scarce resource, especially one designed for ML

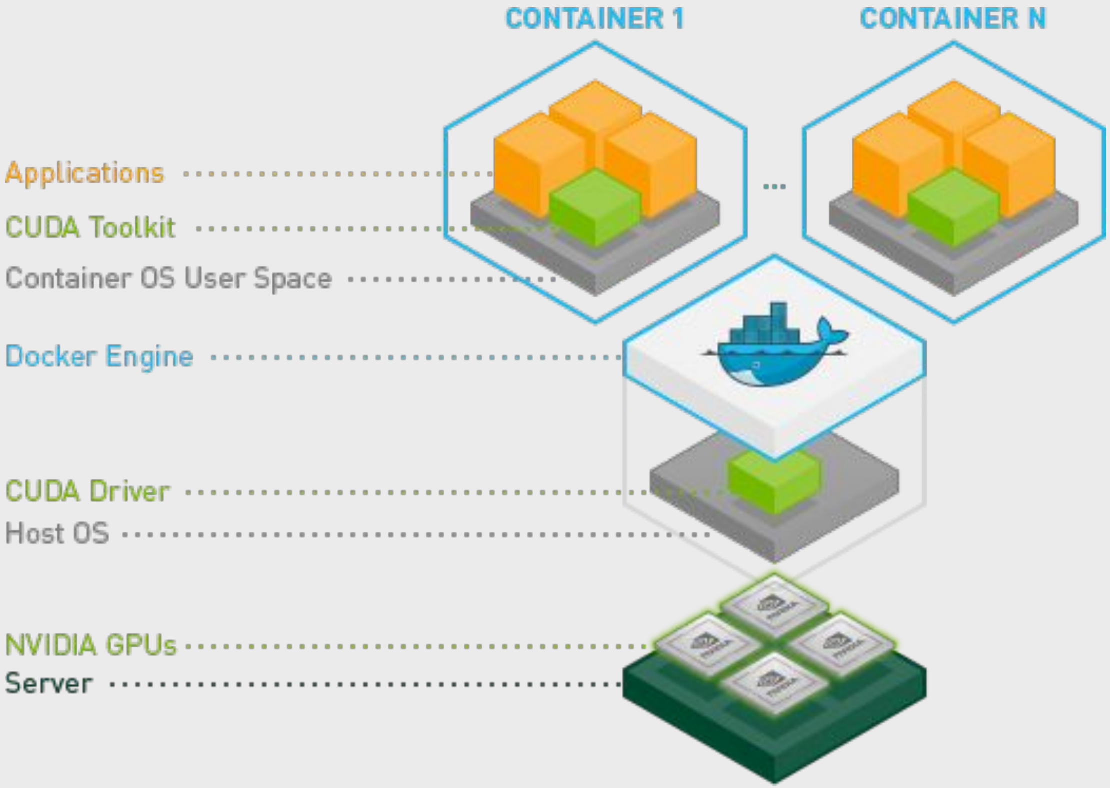
NVIDIA Tesla V100 SXM2

=

NVIDIA Tesla T4 PCIe

=

GPU ON CONTAINER



https://github.com/NVIDIA/nvidia-docker

GPU ON CONTAINER

```
$ docker run --gpus all nvidia/cuda:9.0-base nvidia-smi
$ docker run --runtime nvidia nvidia/cuda:9.0-base nvidia-smi
$ nvidia-docker run nvidia/cuda:9.0-base nvidia-smi
```

```
Mon Sep  2 14:57:28 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.56       Driver Version: 418.56       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  GeForce GTX 105...  Off  | 00000000:01:00.0 Off |                  N/A |
| N/A   48C    P8    N/A /  N/A |      2MiB /  4040MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

DEVICE PLUGINS API HISTORY

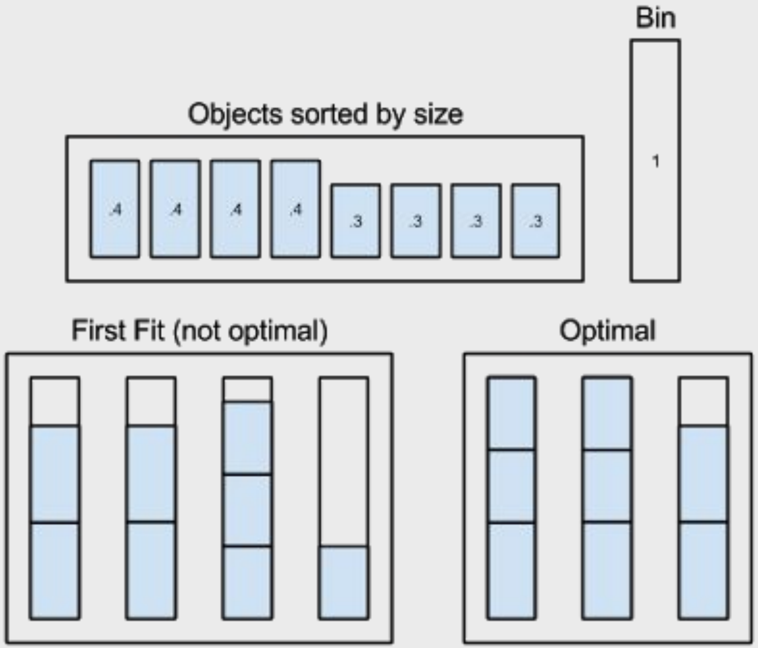
alpha.kubernetes.io/gpu

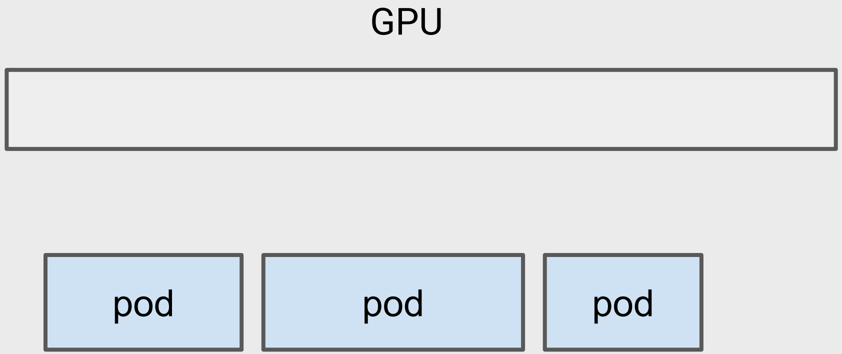
- Deprecated in v1.10, removed 1.11
- Using kernel module and library on host, expose with hostPath
- K8s expose GPU as scheduleable resources

Device Plugins (e.g. nvidia.com/gpu)

- Introduced in v1.8, beta 1.10
- Advertise hardware resources to Kubelet
- Deploy manually or with DaemonSet
- GPUs, high-performance NICs, FPGAs and InfiniBand adapters

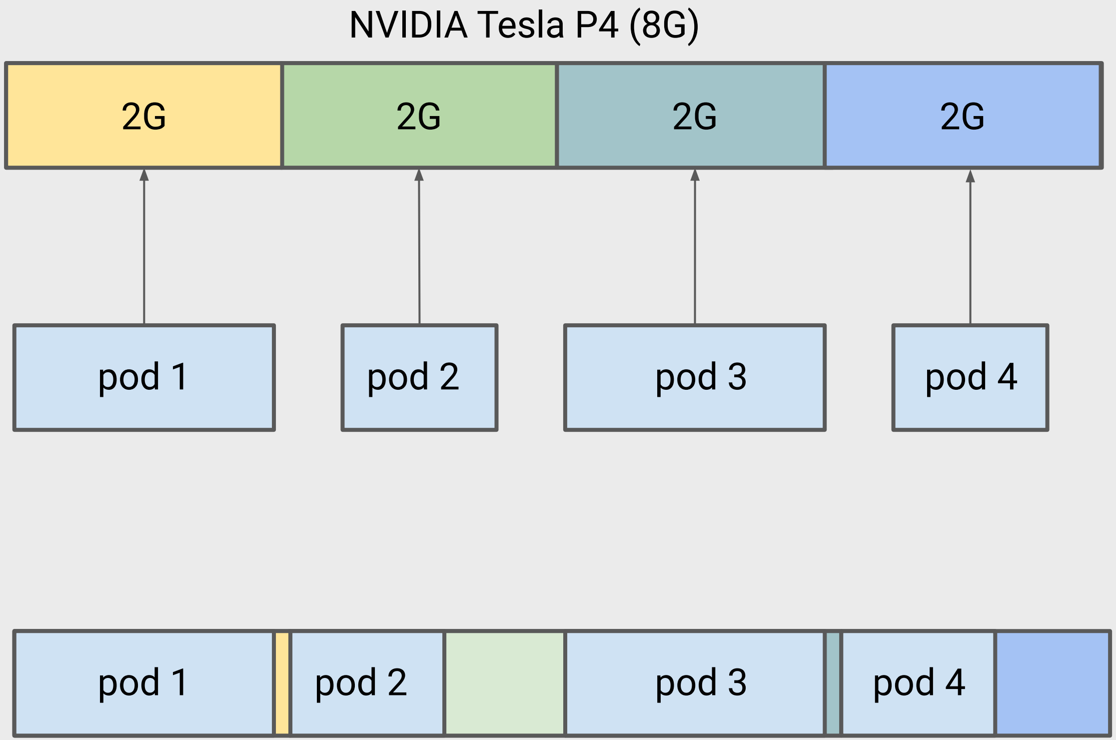
BIN-PACKING ON GPU





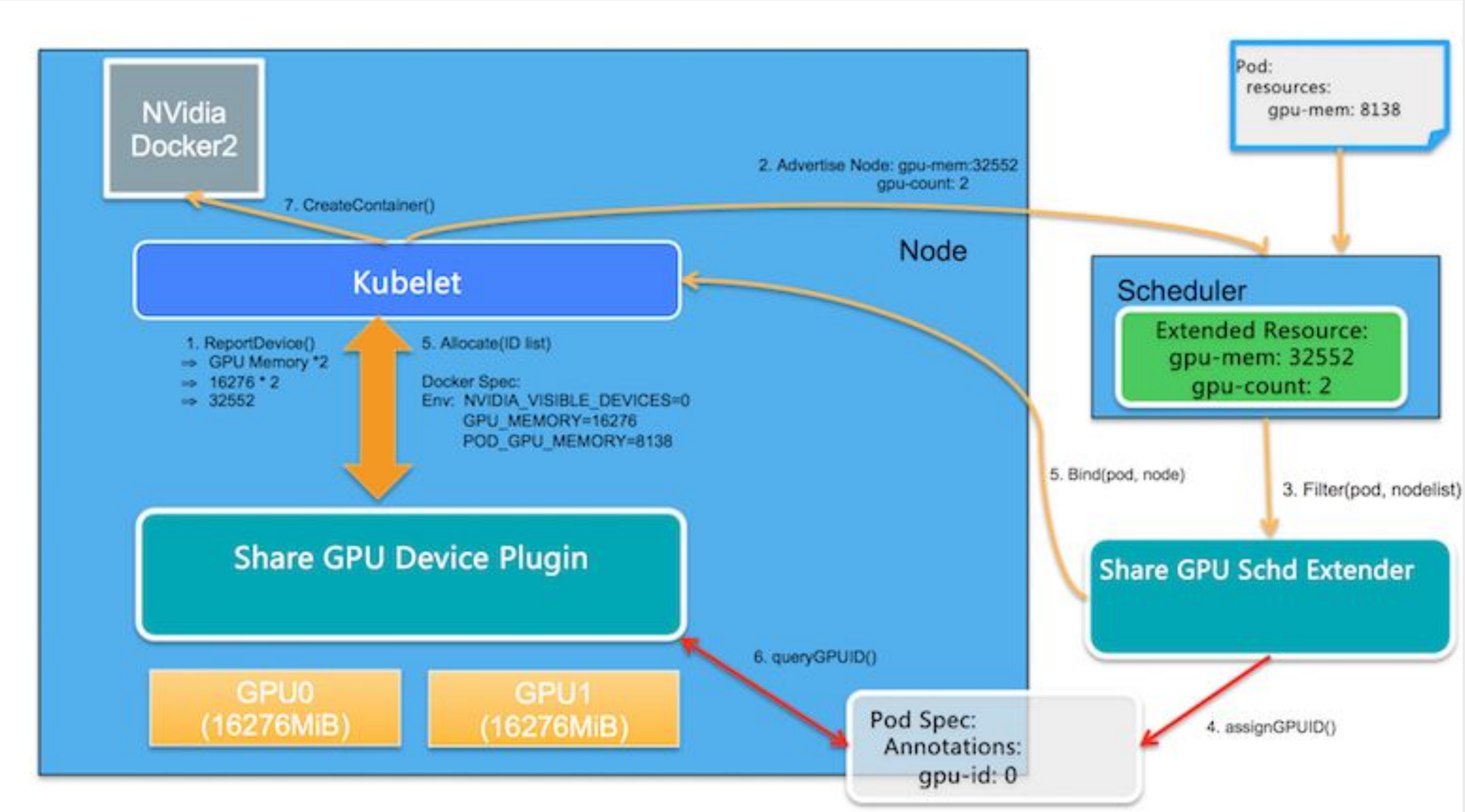
BIN-PACKING ON GPU

Bin-pack based on GPU memory limit



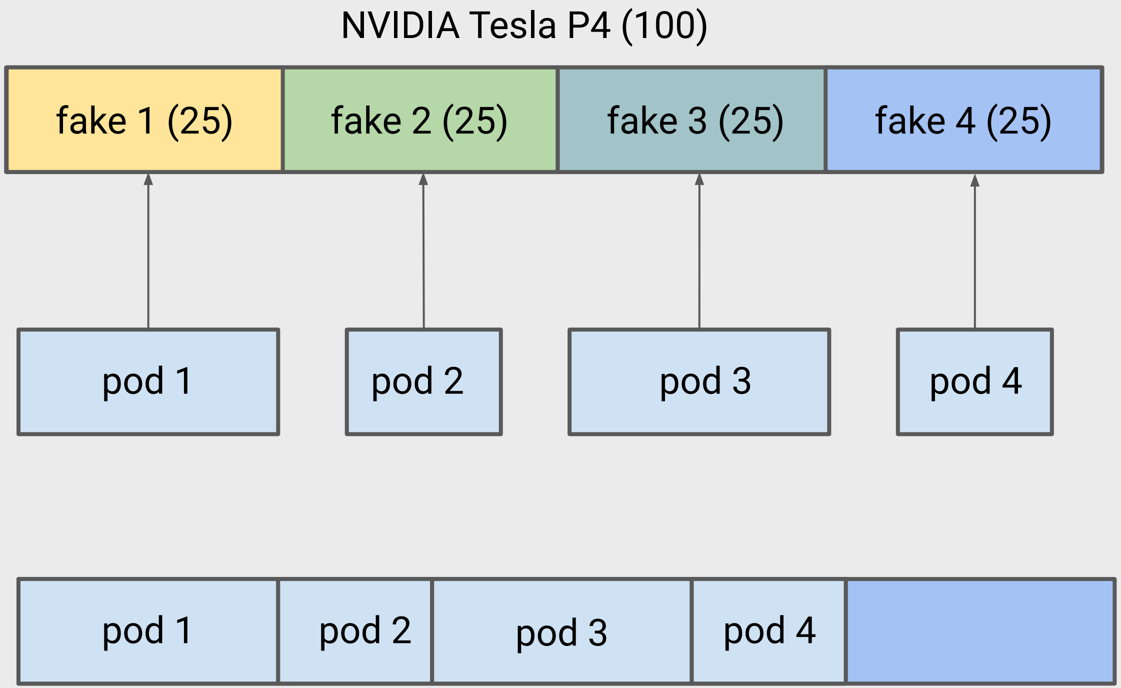
BIN-PACKING ON GPU

https://github.com/AliyunContainerService/gpushare-scheduler-extender

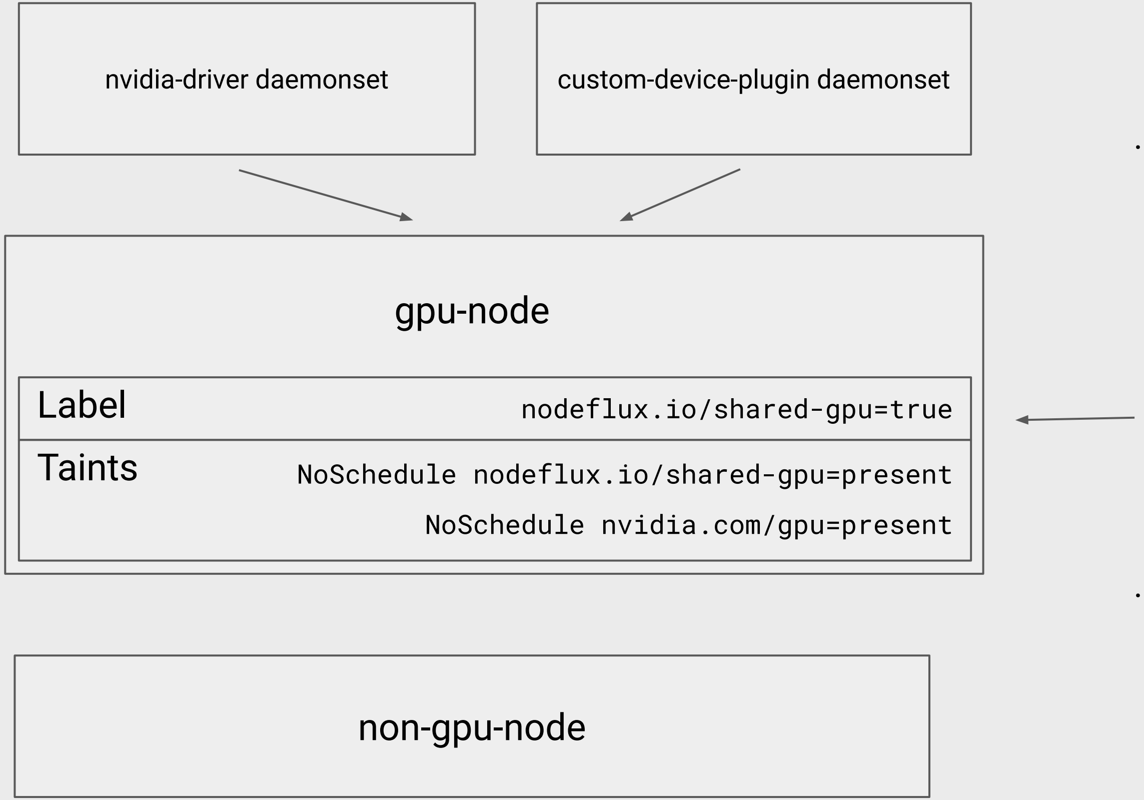


BIN-PACKING ON GPU

Bin-pack based on fake GPU device



BIN-PACKING ON GPU

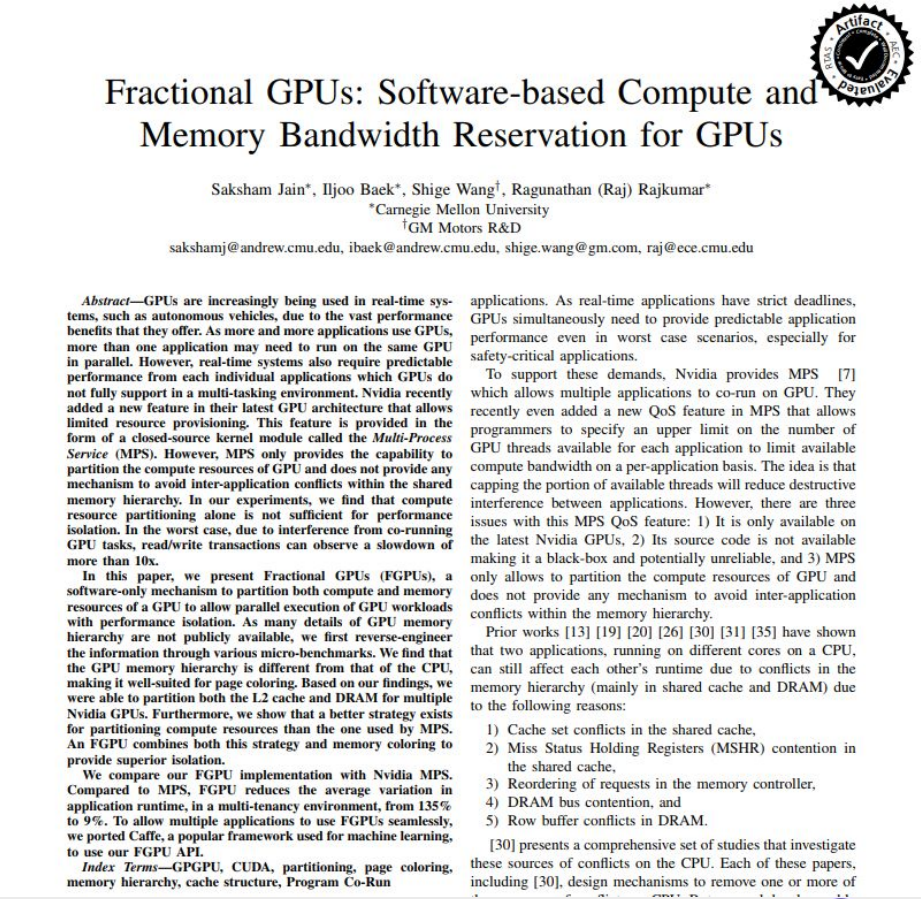



```yaml
...
  spec:
      tolerations:
        - key: nvidia.com/gpu
          operator: Exists
          effect: NoSchedule
      containers:
        - name: cat-detection
          image:
          resources:
            limits:
              cpu: "4"
              memory: "8G"
              nodeflux.io/shared-gpu: 25
...
```

https://github.com/Deepomatic/container-engine-accelerators/

CURRENT HURDLES

- GPU sharing
  (Open K8s Issue #52757)
- Autoscaling support
- A general solution for GPU
- No Minikube support
- No local managed provider



Fractional GPUs: Software-based Compute and Memory Bandwidth Reservation for GPUs

Saksham Jain*, Iljoo Baek*, Shige Wang†, Ragunathan (Raj) Rajkumar*
*Carnegie Mellon University
†GM Motors R&D
sakshamj@andrew.cmu.edu, ibaek@andrew.cmu.edu, shige.wang@gm.com, raj@ece.cmu.edu

Abstract—GPUs are increasingly being used in real-time systems, such as autonomous vehicles, due to the vast performance benefits that they offer. As more and more applications use GPUs, more than one application may need to run on the same GPU in parallel. However, real-time systems also require predictable performance from each individual applications which GPUs do not fully support in a multi-tasking environment. Nvidia recently added a new feature in their latest GPU architecture that allows limited resource provisioning. This feature is provided in the form of a closed-source kernel module called the *Multi-Process Service* (MPS). However, MPS only provides the capability to partition the compute resources of GPU and does not provide any mechanism to avoid inter-application conflicts within the shared memory hierarchy. In our experiments, we find that compute resource partitioning alone is not sufficient for performance isolation. In the worst case, due to interference from co-running GPU tasks, read/write transactions can observe a slowdown of more than 10x.

In this paper, we present Fractional GPUs (FGPUs), a software-only mechanism to partition both compute and memory resources of a GPU to allow parallel execution of GPU workloads with performance isolation. As many details of GPU memory hierarchy are not publicly available, we first reverse-engineer the information through various micro-benchmarks. We find that the GPU memory hierarchy is different from that of the CPU, making it well-suited for page coloring. Based on our findings, we were able to partition both the L2 cache and DRAM for multiple Nvidia GPUs. Furthermore, we show that a better strategy exists for partitioning compute resources than the one used by MPS. An FGPU combines both this strategy and memory coloring to provide superior isolation.

We compare our FGPU implementation with Nvidia MPS. Compared to MPS, FGPU reduces the average variation in application runtime, in a multi-tenancy environment, from 135% to 9%. To allow multiple applications to use FGPUs seamlessly, we ported Caffe, a popular framework used for machine learning, to use our FGPU API.

Index Terms—GPGPU, CUDA, partitioning, page coloring, memory hierarchy, cache structure, Program Co-Run

applications. As real-time applications have strict deadlines, GPUs simultaneously need to provide predictable application performance even in worst case scenarios, especially for safety-critical applications.

To support these demands, Nvidia provides MPS [7] which allows multiple applications to co-run on GPU. They recently even added a new QoS feature in MPS that allows programmers to specify an upper limit on the number of GPU threads available for each application to limit available compute bandwidth on a per-application basis. The idea is that capping the portion of available threads will reduce destructive interference between applications. However, there are three issues with this MPS QoS feature: 1) It is only available on the latest Nvidia GPUs, 2) Its source code is not available making it a black-box and potentially unreliable, and 3) MPS only allows to partition the compute resources of GPU and does not provide any mechanism to avoid inter-application conflicts within the memory hierarchy.

Prior works [13] [19] [20] [26] [30] [31] [35] have shown that two applications, running on different cores on a CPU, can still affect each other's runtime due to conflicts in the memory hierarchy (mainly in shared cache and DRAM) due to the following reasons:

1) Cache set conflicts in the shared cache,
2) Miss Status Holding Registers (MSHR) contention in the shared cache,
3) Reordering of requests in the memory controller,
4) DRAM bus contention, and
5) Row buffer conflicts in DRAM.

[30] presents a comprehensive set of studies that investigate these sources of conflicts on the CPU. Each of these papers, including [30], design mechanisms to remove one or more of

https://ieeexplore.ieee.org/document/8743200

EXAMPLE ON GKE

https://github.com/eufat/presentasi-k8s

Thank you

Psst, we're hiring!

www.nodeflux.io