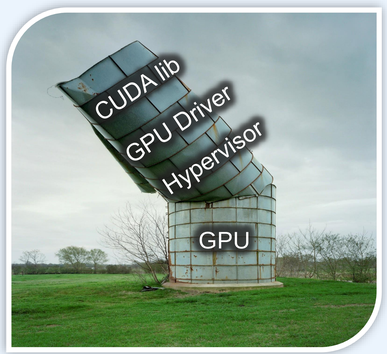
NIGHTWATCH: Remoting Accelerator APIs through the Hypervisor

Hangchen Yu, Amogh Akshintala, Arthur Peters, Christopher J. Rossbach

The University of Texas at Austin · University of North Carolina at Chapel Hill · VMware Research Group

Accelerator Stacks are Silos

- Hardware Interface: MMIO, mmap'd command queues
- Software Interface: vendor-specific drivers, proprietary protocols



Interposition **only** possible at the top or bottom of silo

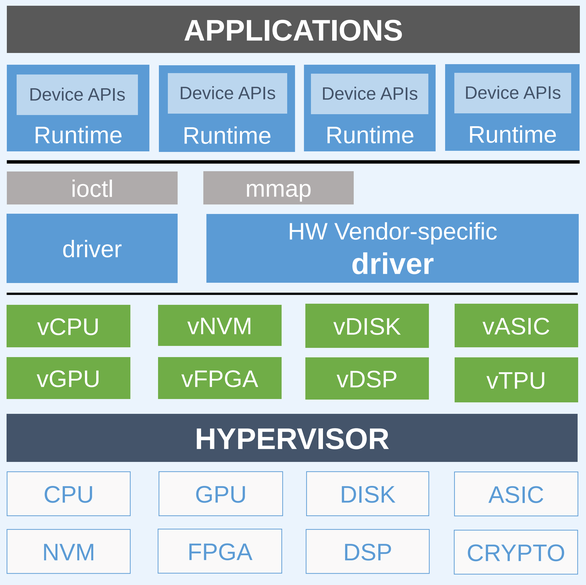
Silos Complicate Virtualization

API forwarding: sacrifices interposition and compatibility

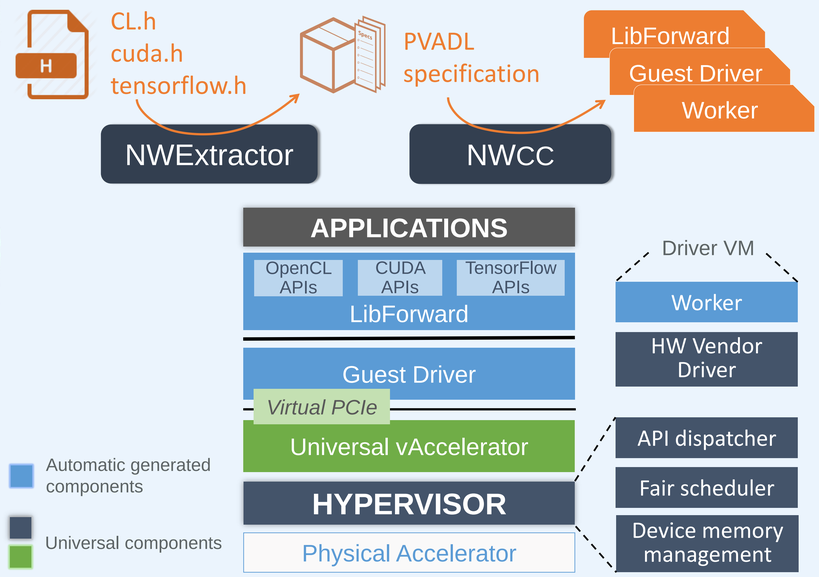
Para-virtual I/O: e.g. SVGA translates guest interactions into DirectX which leads to serious complexity and compatibility issues

Full-virtualization: causes significant overheads by trap-based interposition

SRIOV: remains lacking by hardware support (< 0.95% NVIDIA GPUs)



NIGHTWATCH: Automatic Accelerator Virtualization



Automatic Generation

Development Effort

	# of API	Lines of Spec	LOC (generated)	Time
OpenCL	88	4,200	4,150	A handful of days
CUDA	211	6,350	8,150	
TensorFlow	160	5,350	6,900	
MVNC API	25	910	2,450	

Example

```
CUresult CUDAAPI cuMemcpyHtoD(
        CUdeviceptr dstDevice,
        const void* srcHost,
        size_t byteCount)
```

```
cuMemcpyHtoD:
    async: False
    args:
        dstDevice:
            - type: Cudeviceptr
        srcHost:
            - type: const void*
            - dim: 1
            - length: byteCount
        byteCount:
            - type: size_t
        ret:
            - type: Curesult
```

LibForward

```
CUresult CUDAAPI cuMemcpyHtoD(
        CUdeviceptr dstDevice,
        const void* srcHost,
        size_t byteCount)
{
        INIT_CUDA_PARAM(param);
        param.base.api_id = CU_MEMCPY_H_TO_D;
        // set arguments
        param.arg1_0 = dstDevice;
        param.arg2_0 = byteCount;
        param.arg3_0 = srcHost;
        // compute data size
        param.base.dpool_size += COMPUTE_SIZE(
                param.arg3_0, byteCount);
        // forward call to guest driver
        IOCTL_TO_DRIVER(&param);
        return param.ret_arg0;
}
```

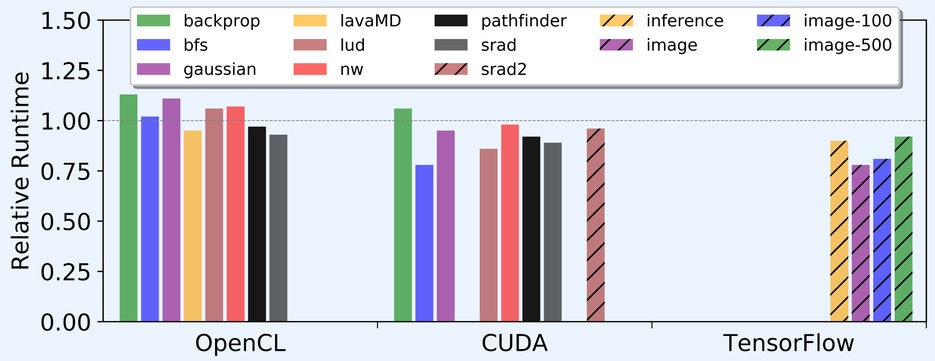
Guest Driver

```
// marshal data to accelerator shared buffer
switch (guest_base->api_id) {
case CU_MEMCPY_H_TO_D:
        COPY_FROM_GUEST(arg1_0);
        COPY_FROM_GUEST(arg2_0);
        COPY_SPACE_FROM_GUEST(arg3_0,
                guest->arg2_0);
        break;
}
// copy result from accelerator shared buffer
switch (guest_base->api_id) {
case CU_MEMCPY_H_TO_D:
        copy_to_user(&arg->ret_arg0,
                &host->ret_arg0, sizeof(CUresult));
        break;
}
```

Worker

```
switch (param->base.api_id) {
case CU_MEMCPY_H_TO_D:
        const void* srcHost = GET_PTR_FROM_DPOOL(
                param->arg3_0,
                const void*);
        param->ret_arg0 = cuMemcpyHtoD(
                param->arg1_0,
                srcHost,
                param->arg2_0);
        break;
}
```

Evaluation



- **Compatibility recovered: stack generation is automatic**
- **Interposition recovered: APIs forwarded over VMM-managed transport**
- **Near-native performance**
- Scales to 16 VMs
- Fair scheduling with heterogeneous workloads
- Migration overhead < 40 ms
- Slowdown under memory over-subscription < 2.5×