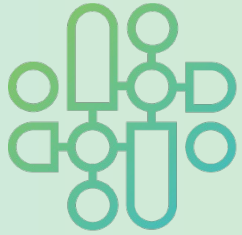


SC23
Denver, CO | i am hpc.

GPU Acceleration in Unikernels Using Cricket GPU Virtualization

Niklas Eiling, Martin Kröning, Jonathan Klimt, Philipp Fensch, Stefan Lankes

RWTH Aachen University

Funded by
the European Union

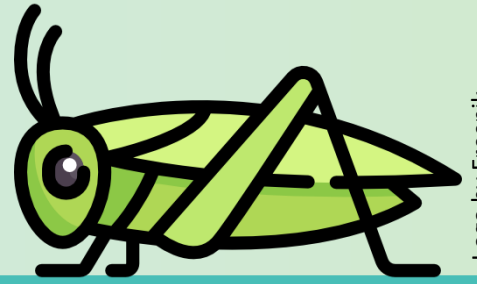
Motivation

Status Quo

- GPUs are crucial for HPC
- Unikernels are emerging (Cloud)
  - Convergent Computing
  - Lower overhead
  - Lower complexity

Goal

- Let’s use GPUs from Unikernels!
- Challenge: GPU drivers are proprietary and complex
  - Driver porting for Unikernels is infeasible
- Sharing of GPUs is difficult



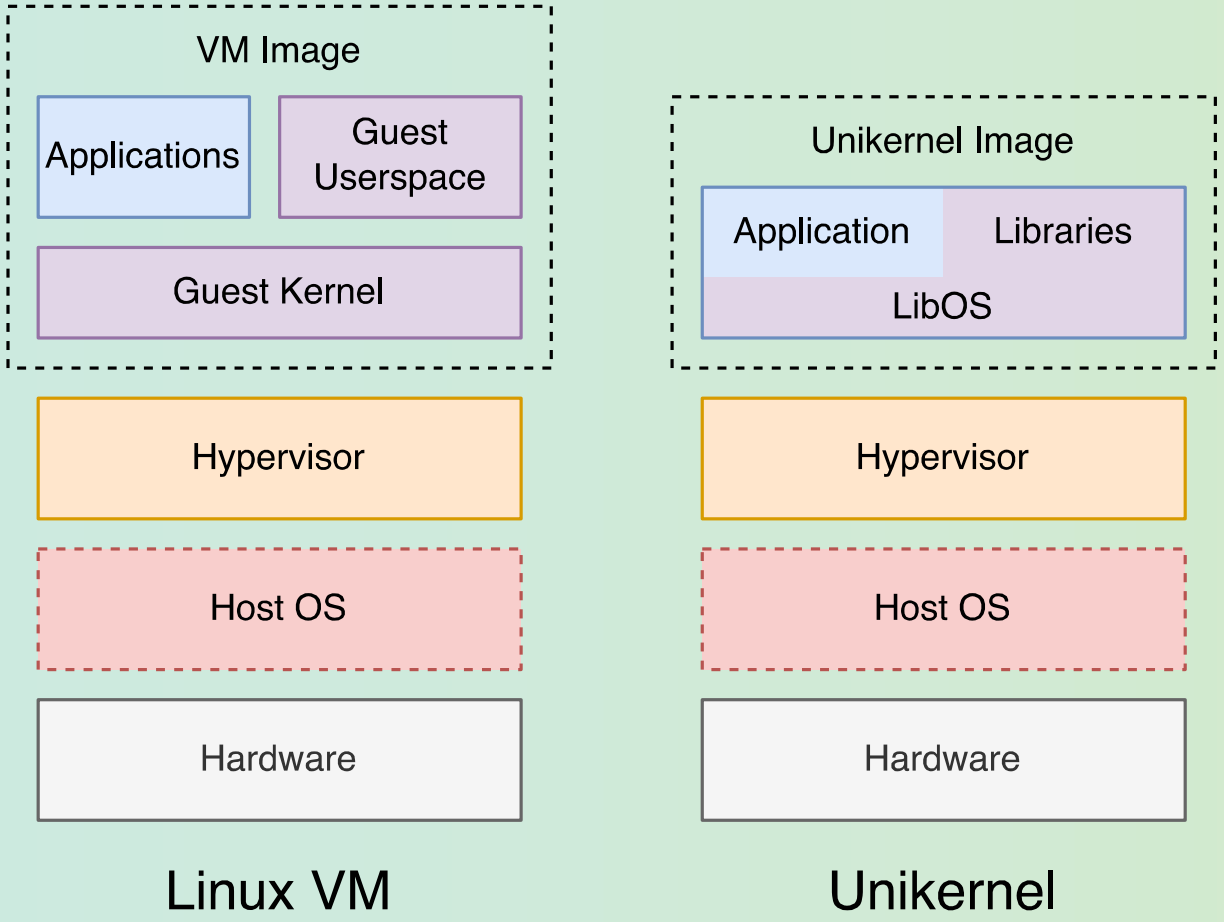
Logo by Freepik

Solution

- Port Cricket GPU Virtualization

Unikernels

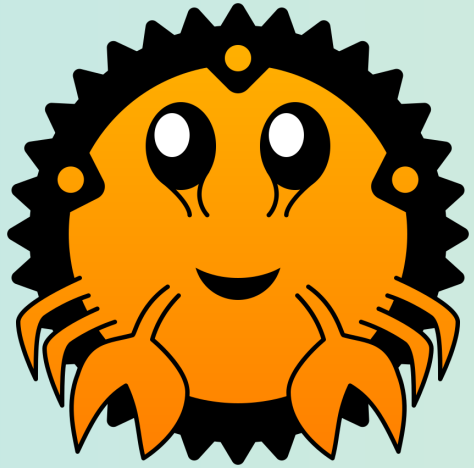
- Specialized for use cause
  - Tiny images
- One process per image
  - No isolation necessary
- Single address space operating system
  - No address space context switch
- Single privilege level
  - No privilege context switch
- System calls are just function calls



Unikernel Selection

Hermit

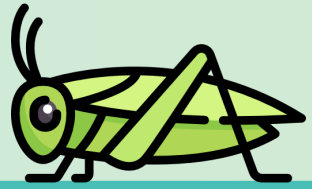
- In-House Solution
- Written in Rust
- Custom Syscall Interface



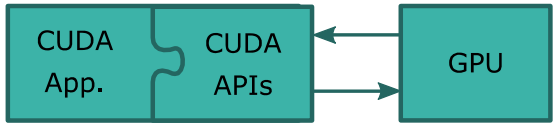
Unikraft

- Commercially Successful
- Written in C
- Linux Syscall Interface

Cricket: A Virtualization Layer for GPUs



API Remoting



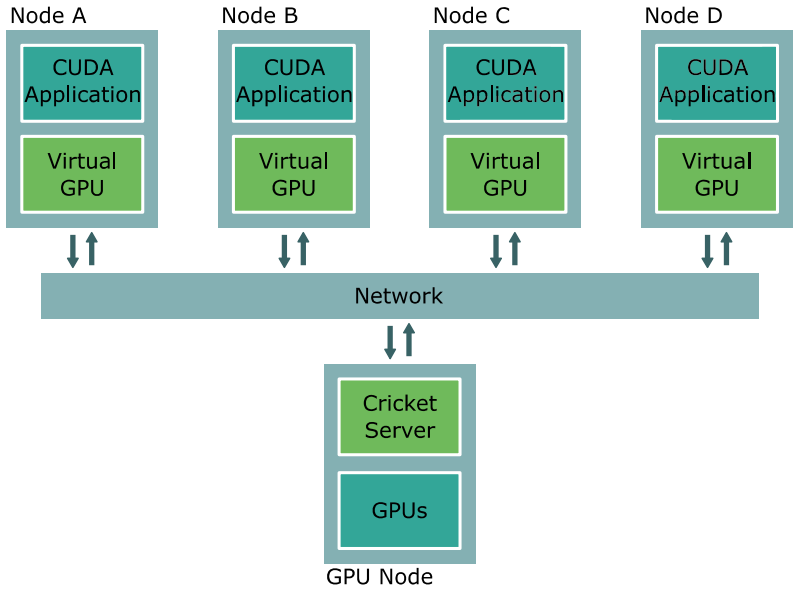
(a) GPU application without virtualization

(b) GPU application with virtualization layer

- Separates proprietary device dependent code into separate process
- Allows full control of device interactions
  - Control, manipulation and limiting the use of GPU resources
- Low virtualization overhead

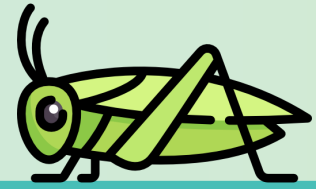
Use Cases

Remote execution, Scheduling, Monitoring



Niklas Eiling, Stefan Lankes, and Antonello Monti
An Open-Source Virtualization Layer for CUDA Applications (2020)

Adapting Cricket for Unikernels

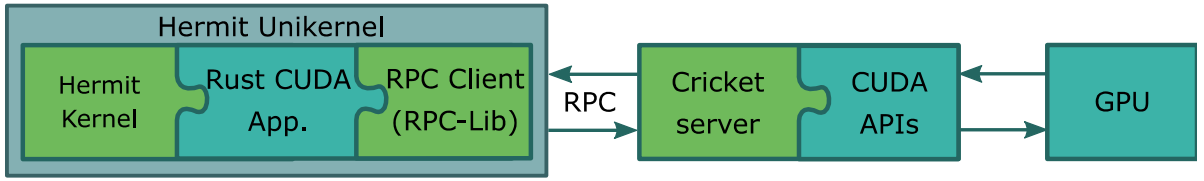


API Remoting

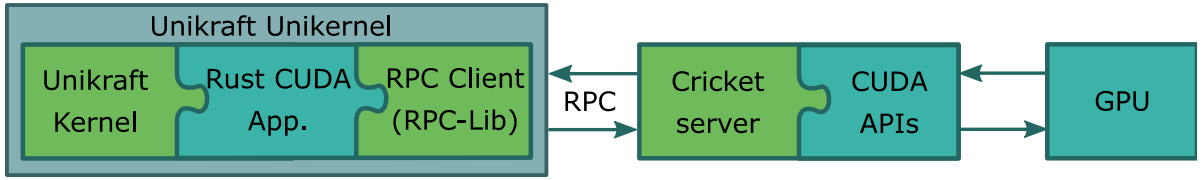
- Cricket is based on ONC RPCs
- Open-Source C Implementation is old and complex
- For Unikernels: New Rust implementation of RPC client
  - Simpler OS Interface than C Impl.
  - Hermit: Client side is rust-only
- All user code is run inside Unikernel
- Only CUDA APIs are run outside

Cricket for Unikernels

(a) Cricket with C client



(b) Cricket with Hermit client



(c) Cricket with Unikraft client

Rust Procedural Macros Generate Remote Procedure Calls

Definition in RPCL

```
struct MyType {
    int x;
    int y;
};

program PROG {
    version VERS {
        MyType PROC_1(int, float) = 1;
    } = 1;
} = 10050;
```

```
#[include_rpcl("rpc_cuda.x")]
struct RPCConnection;
...
let rpc_connection =
RPCConnection::new("137.226.133.199");
...
rpc_connection.rpc_cusolverdndgetrs(...);
```

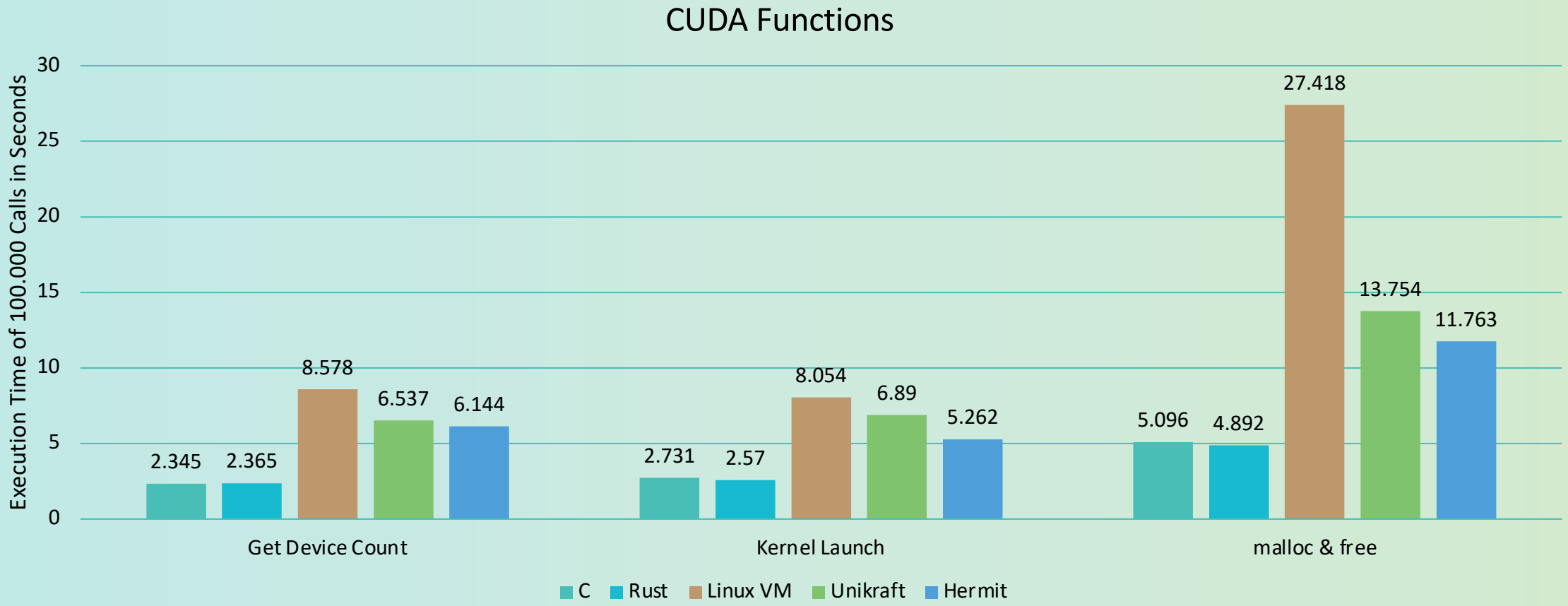
CUDA in Rust Example

```
let mut client = client();
let mut host_mem = buf(size);
let mut device_mem = client.cuda_malloc(size as u64).unwrap();
client
    .cuda_memcpy_htod(&host_mem, &mut device_mem)
    .unwrap();

  • • •

client.cuda_free(device_mem).unwrap();
```

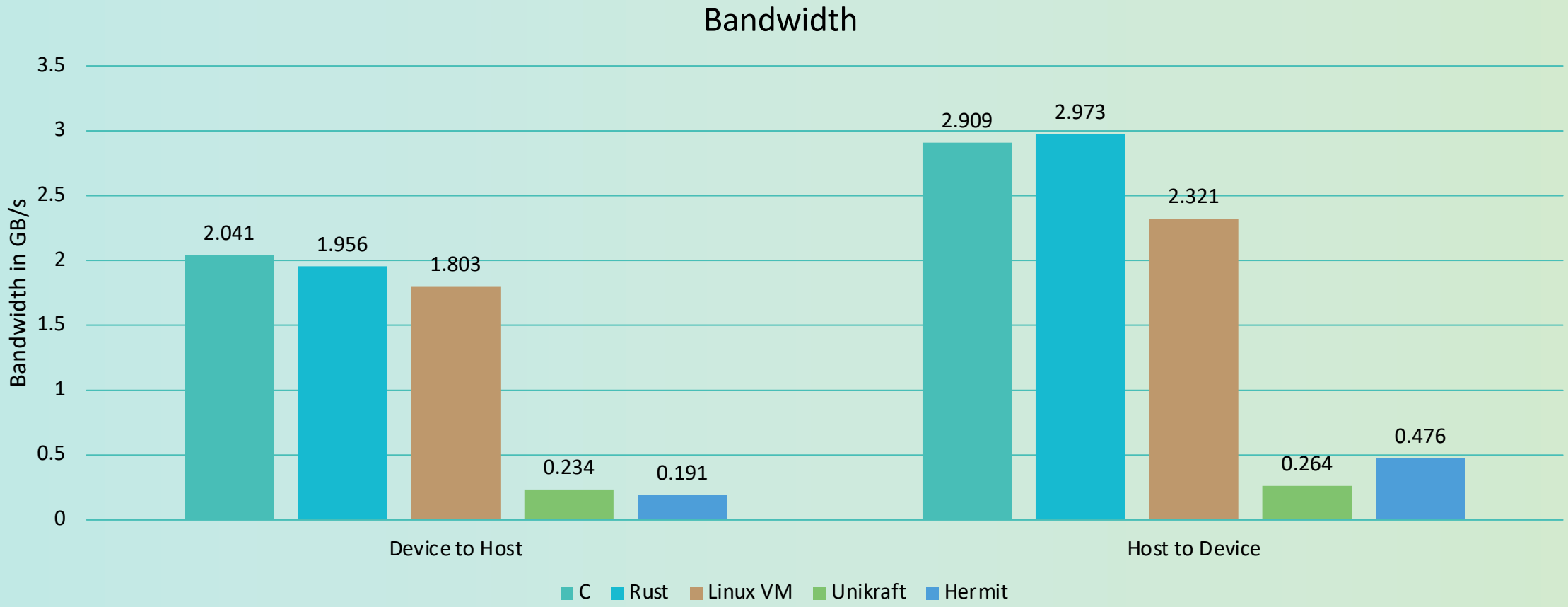
Benchmark Setup

Name	Application Lang.	OS	Hypervisor	Network
C	C	Rocky Linux	-	Native
Rust	Rust	Rocky Linux	-	Native
Linux VM	Rust	Fedora VM	QEMU/KVM	virtio
Unikraft	Rust	Unikraft	QEMU/KVM	virtio
Hermit	Rust	Hermit	QEMU/KVM	virtio

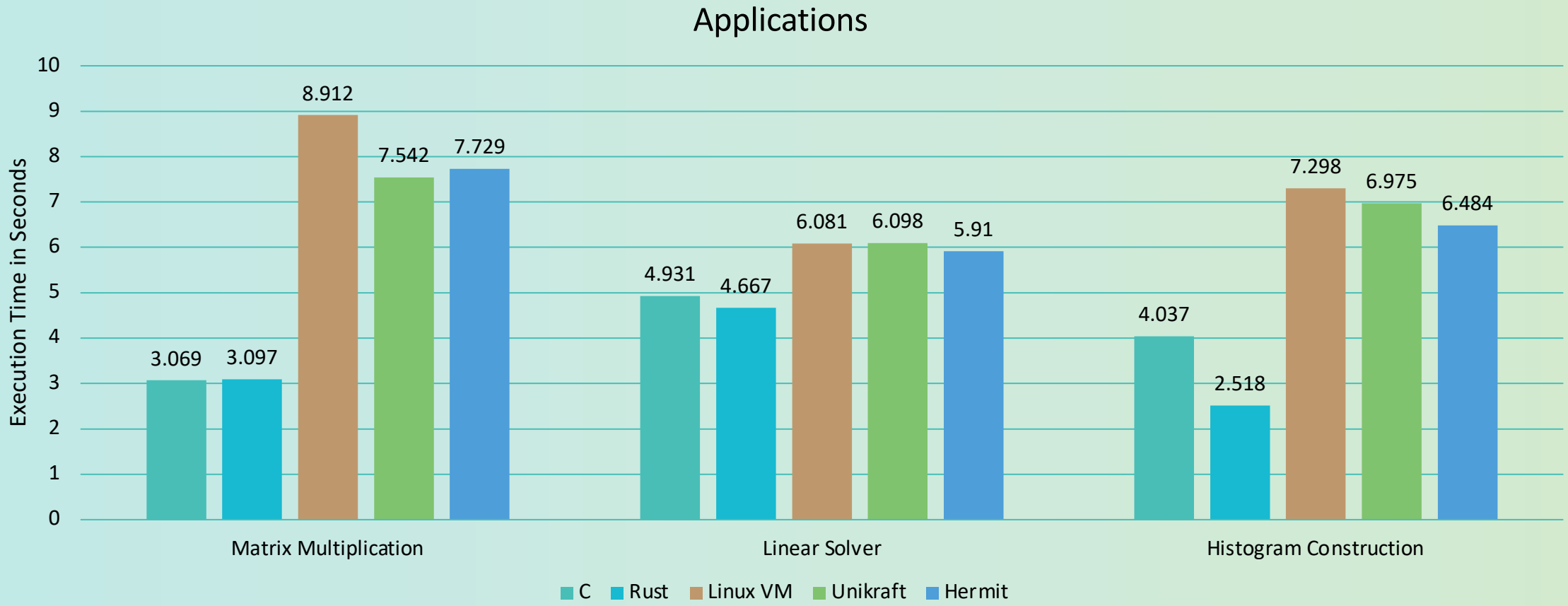
Microbenchmarks



Microbenchmarks



CUDA Samples Benchmarks

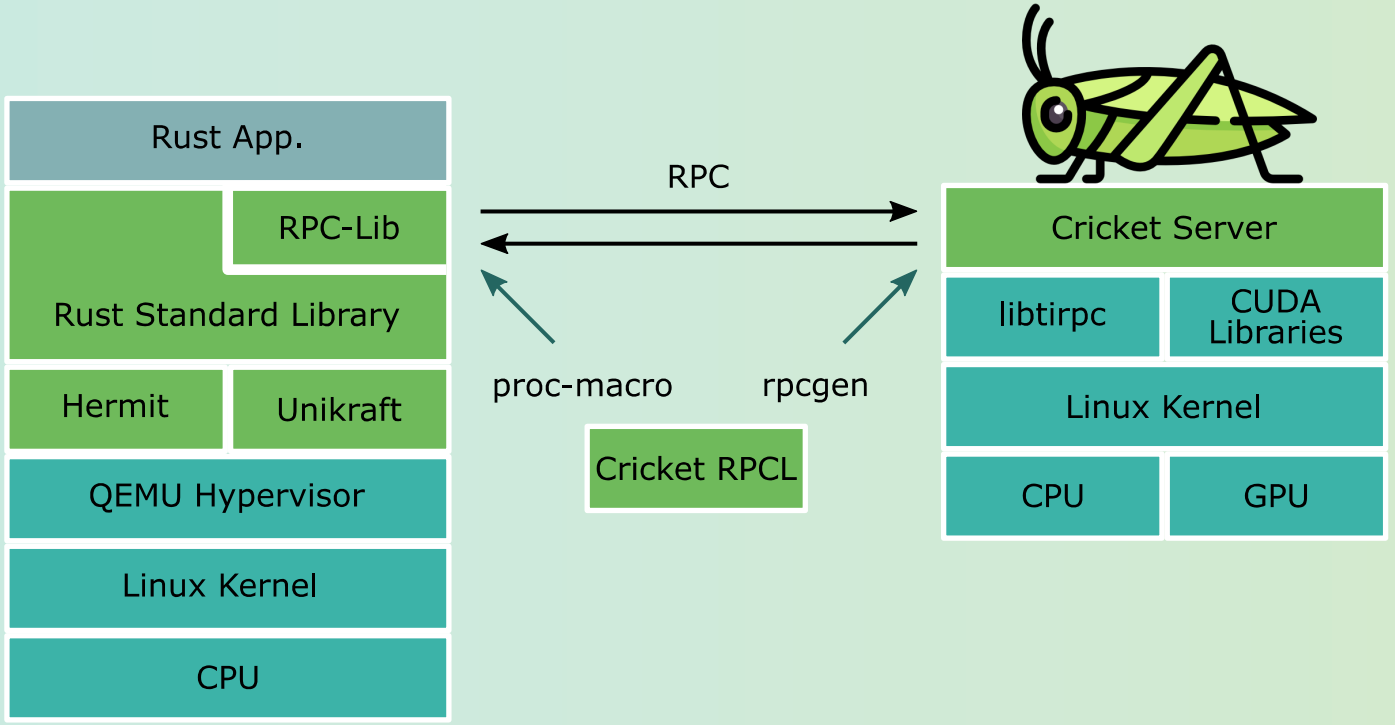


Conclusion

- RPC based GPU access is feasible
- Low Bandwidth with Unikernel Networking

Future Work

- Implement Virtio-net Multiqueue
- Parallelize Network Stacks
- DPDK for Host Kernel Bypass?
- vDPA for Full Host Bypass?



Questions?