

Programming Across the VM Boundary with eBPF: Lessons from Revisiting Phantom Tracker

Himadri Chhaya-Shailesh^{1, 2, 4} Roy Nchang³ Andrea Righi⁴

¹Whisper, Inria Paris

²KrakOS, Grenoble INP - UGA, Inria Grenoble

³GSoC Intern, National Advanced School of Engineering (ENSPY), Yaoundé, Cameroon

⁴GSoC Mentors

06/10/2026 | LPC'26 eBPF Track

Agenda

Context

Background & Motivation

Prior Discussions

GCC GSoC'26: Revamped Phantom Tracker using BPF

BPF-compatible Para-virtualized IVSHMEM Driver Overview

Next Steps

References

Context

A Brief History of Phantom Tracker

1. **PhD Thesis:**

- Student: Himadri Chhaya-Shailesh
- Supervisors: Jean-Pierre Lozi & Julia Lawall
- Defended on 17 December, 2025

2. **GCC GSoC Project:**

- Intern: Roy Nchang
- Mentors: Thomas Schwinge, Tobias Burnus, Andrea Righi, and Himadri Chhaya-Shailesh
- Deliverables: BPF implementation of thesis prototype + patches for libgomp

BPF Usage Across Three Layers

1. **Parallelization**

→ Tracking and managing OpenMP worker threads in the userspace.

2. **Scheduling**

→ Tracking task scheduling decisions across the host and guest schedulers.

3. **Para-virtualization**

→ Enabling para-virtualized shared memory communication for QEMU/KVM.

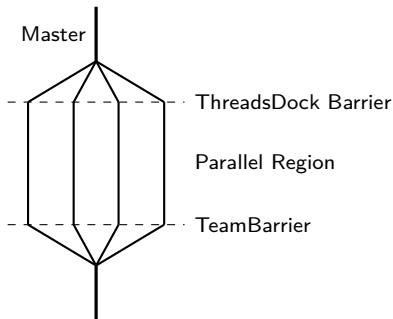
Background & Motivation

OpenMP Execution in Cloud VMs

- High-performance computing (HPC) relies on parallel execution.
- OpenMP is widely used for shared-memory parallelization.
- Running OpenMP applications in cloud VMs is commonplace.
- Oversubscription is a popular cost-cutting practice in cloud deployments.
- OpenMP performance heavily relies on **task scheduling** and **barrier synchronization**.
- The corresponding default choices in `libgomp` are static and virtualization-oblivious.

OpenMP Fork-Join Execution Model

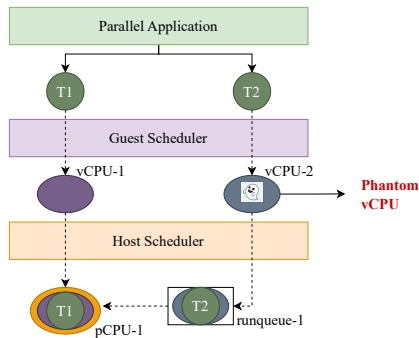
- OpenMP creates a team of worker threads for each parallel region, with each thread executing a share of the parallel work.
- Threads synchronize at barriers: each thread waits until all threads in the team have reached the barrier before execution proceeds.



Phantom vCPUs Under Oversubscription

A vCPU is **phantom** if:

1. It is waiting in the runqueue of a pCPU on the host.
2. A guest task running on this vCPU is stalled because the vCPU itself is not executing.



Thesis: Scheduler Guided OpenMP Execution in Cloud VMs

Improve OpenMP performance in oversubscribed Linux VMs by:

- Combining task-scheduling insights from the host and guest schedulers.
- Using these insights to guide OpenMP runtime decisions in the guest regarding:
 - Degree of Parallelism (DoP) for each parallel region.
 - Barrier synchronization mechanism at each barrier.

Thesis Contributions

1. Enable communication between the host and guest schedulers.
2. Phantom Tracker: An algorithm that tracks vCPU states on the host and computes a time-weighted average, `phantom_average`, capturing how host overload affects the scheduling of the target VM's vCPUs.
3. Use `phantom_average` to guide OpenMP execution in the guest.

Thesis Prototype Limitations

1. Requires patching both the host and guest Linux kernel schedulers.
2. Requires compiling and running custom kernels on both the host and guest.
3. Relies on ad hoc drivers and kernel headers for the QEMU IVSHMEM device-based shared memory communication between the host and guest schedulers.
4. Heavily modifies `libgomp` for implementing phantom-aware policies.

Prior Discussions

Previously Presented At

- Linux Plumbers Conference (LPC) 2024¹
- Kernel Recipes 2024²
- Free and Open Source Software Developers' European Meeting (FOSDEM) 2026³
- OpenMP Affinity Telco Workgroup Meeting 2026⁴

¹A case for using para-virtualized scheduling information with sched_ext schedulers

²Dual level of task scheduling for VM workloads – Pain points & Solutions

³Libgomp Optimizations for Scheduler Guided OpenMP Execution in Cloud VMs

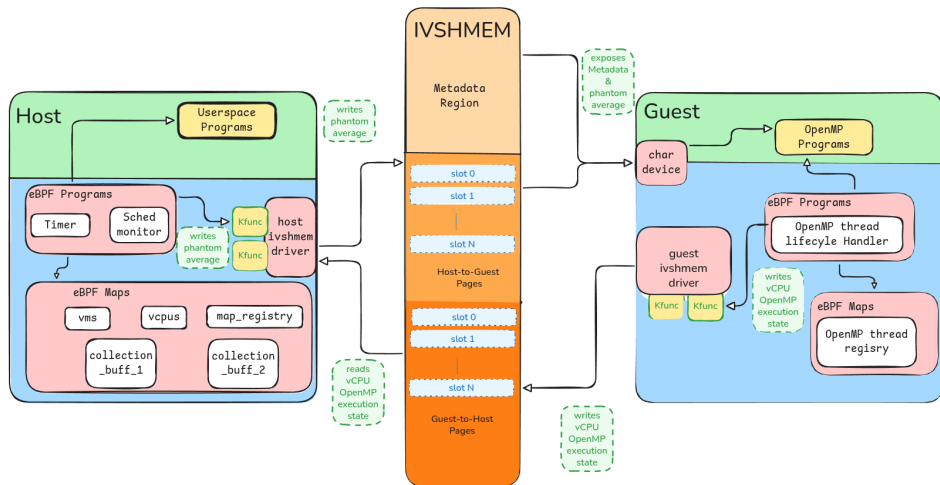
⁴[omp-pvsched] Linux Task Scheduler Guided OpenMP Execution in Oversubscribed Cloud VMs

GCC GSoC'26: Revamped Phantom Tracker using BPF

Revisiting Phantom Tracker

1. BPF-compatible para-virtualized shared-memory drivers for QEMU IVSHMEM.
2. Lightweight modifications to `libgomp` that exploit recent BPF features.

Revised Architecture of the Phantom Tracker



Computing phantom_average on the Host

Periodic BPF timer firing every $1/f_{\text{clock}}$ (4 ms on a 250 Hz system)

For each registered VM, Phantom Tracker performs these steps:

1

BPF SchedMonitor

Collect phantom samples in BPF map-backed buffers at `sched_switch` and `sched_wakeup` events.

2

BPF Timer

Compute the time-weighted rolling `phantom_average` from the collected phantom samples whenever the timer fires.

3

BPF-Compatible IVSHMEM Access

Read guest scheduling decisions for OpenMP threads from the guest-to-host message area, and write `phantom_average` to the host-to-guest message area.

Using `phantom_average` in the Guest

For the target OpenMP application running inside the guest, Phantom Tracker and `libgomp` perform these steps:

4

BPF-Compatible IVSHMEM Access

Write OpenMP thread scheduling decisions to the guest-to-host message area, and read `phantom_average` from the host-to-guest message area.

5

Phantom-Aware Barrier Synchronization

Decide whether each thread should continue spinning or switch to blocking at each barrier using BPF USDT.

6

Phantom-Aware Degree of Parallelism

Dynamically determine how many OpenMP worker threads to use at the beginning of each parallel region using BPF USDT.

BPF-compatible Para-virtualized IVSHMEM Driver Overview

Design Criteria

- **Minimal Access Latency:** Data may need to be recorded at every scheduler switch, requiring low-latency shared-memory access in both the host and the guest.
- **BPF-compatible Access:** Data produced by eBPF programs on both the host and the guest must be directly accessible through shared memory.
- **VM Lifecycle Awareness:** The host must dynamically attach and detach per-VM para-virtualized shared memory as VMs are created and destroyed.

Existing Technologies

	Shared Memory	BPF Compatibility	Limitations
IVSHMEM	✓	×	Designed for userspace shmem; fixed and static backend size with kvmfr.
virtio	×	×	Virtqueue and notification overhead; no direct shared-memory fast path.
vsock	×	×	Socket-based communication with per-VM CID addressing; no direct memory access.
BPF Arena maps	×	✓	Accessible within a kernel/userspace domain; no host–guest sharing across the VM boundary.

BPF-Compatible Driver Design

Challenges:

- BPF programs cannot write directly to arbitrary kernel memory.
- eBPF map memory is allocated dynamically when the map is created and is not discoverable across host-guest kernel boundaries.

Our Solution: IVSHMEM Host-Guest Drivers Exposing kfuncs

- Our Drivers implement ordinary device operations that perform direct reads and writes to the mapped IVSHMEM region.
- These operations are exposed to Phantom Tracker's host-guest BPF programs as kfuncs.
- Phantom Tracker's host-guest BPF programs invoke these kfuncs to access the IVSHMEM device memory from the scheduler trace points in the host and the guest kernels as well as from the USDTs defined in guest userspace `libgomp`.

Synchronizing VM and IVSHMEM Device Lifecycles

- Each target VM is assigned one IVSHMEM device at boot, and the device is removed when the VM shuts down.
- Phantom Tracker's data resides in this device-backed memory, and Phantom Tracker further uses device events to track VM lifecycle.
- Device creation and removal therefore correspond to VM arrival and departure from Phantom Tracker's perspective.
- No separate VM lifecycle management mechanism is required.
- The host IVSHMEM driver manages such per-VM devices and handles their creation and removal events.

Next Steps

Next Steps

- Discuss the broader applicability of our BPF-compatible host–guest IVSHMEM drivers at LPC'26.
- Prepare patches to upstream these drivers to the Linux kernel, enabling us to move forward with upstreaming phantom-aware policies in `libgomp` and proposing phantom-aware extensions to the OpenMP standard.
- Work toward a generic, programmable design that can serve as a backend for BPF Arena maps, allowing Phantom Tracker-specific shared-memory details to move from the drivers into our BPF programs.

References

References

Links:

- **Himadri's PhD Thesis:** Available on HAL, a multidisciplinary open archive
- **Roy's GSoC Project:** Project details
- **Andrea's Patches for Backing Shared Guest RAM with BPF Arenas:**
 - [PATCH 1/2] bpf: Allow pinned arenas to back shared guest RAM
 - [PATCH 2/2] selftests/bpf: Test two-way arena access with two guests
- **Our Work-in-Progress BPF-Compatible Host-Guest IVSHMEM Drivers:**
Available in the Phantom Tracker GitHub repository

Contacts:

- **Himadri Chhaya-Shailesh:** himadrics@protonmail.com
- **Roy Nchang:** royfru44@gmail.com
- **Andrea Righi:** arighi@nvidia.com