

EBPF '26 · PRAGUE, CZECH REPUBLIC

NCCLbpf: Verified, Composable Policy Execution for GPU Collective Communication

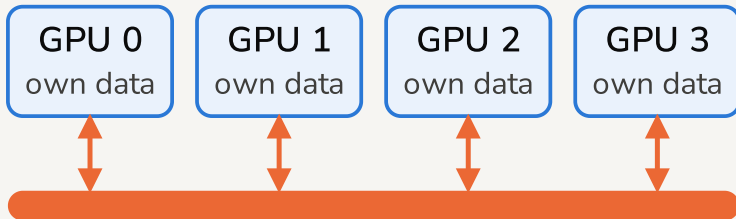
Yusheng Zheng

github.com/eunomia-bpf/nccl-eBPF

Every step of LLM training and serving waits on NCCL

NCCL: NVIDIA's GPU communication library, under PyTorch, Megatron-LM, DeepSpeed, inference engines, and HPC/MPI programs.

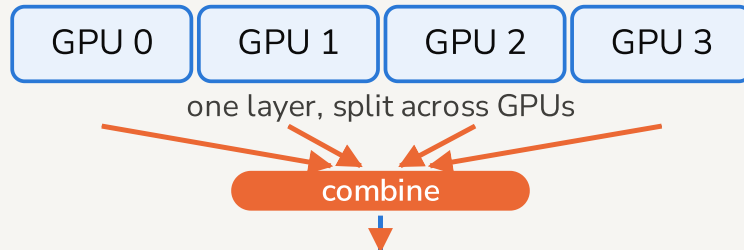
Training: every step



exchange gradients, then the next step

Llama 3: 16,384 GPUs in one job.

Inference: every layer, every token



next layer, then the next token

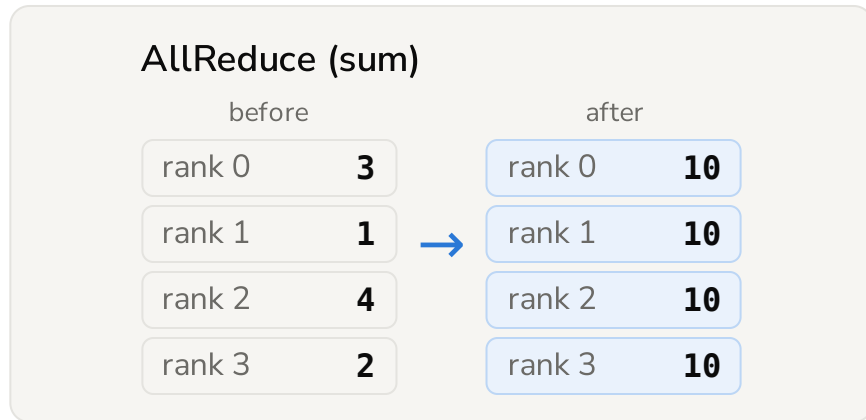
Small messages, on the latency-critical path.

A collective: a group of GPUs exchanges data in one call

A group of processes communicates together. Each process is a rank, usually one per GPU; the group is a communicator.

Every rank calls the same operation, unlike point-to-point `send / recv` between two processes.

The most common one is **AllReduce**: combine every rank's data (e.g., sum it) and give the result to every rank.



Broadcast

before	after
r0 A	r0 A
r1 -	r1 A
r2 -	r2 A
r3 -	r3 A

one rank's data to all

AllGather

before	after
r0 A	r0 ABCD
r1 B	r1 ABCD
r2 C	r2 ABCD
r3 D	r3 ABCD

everyone gets every part

ReduceScatter

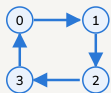
before	after
r0 a	r0 Σ_0
r1 b	r1 Σ_1
r2 c	r2 Σ_2
r3 d	r3 Σ_3

sum, then each rank keeps one part

One call, many implementations

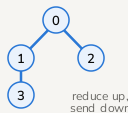
The caller says "AllReduce"; the library decides how. No choice wins everywhere: it depends on message size and hardware.

Algorithm: Ring



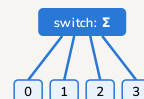
Pass chunks around a ring.
Suits large messages.

Algorithm: Tree



Combine up a tree, send the
result down. Suits small
messages.

Algorithm: NVLS



The NVLink switch adds the
data up itself (NVLink SHARP).

Also: protocol and channels

Protocol: how data is packed, low-latency for small
messages or high-bandwidth for large ones. **Channels:**
how many parallel lanes.

Every rank must choose the same way

Each algorithm fixes who sends what to
whom. Rank 0 on Ring, rank 1 on Tree $\Rightarrow$
each waits for data that never comes: **the job
hangs.**

NCCL: where these choices are made

↓ `ncclAllReduce(buf, count, sum, comm)`

NCCL on each rank, for every collective call

choose algorithm · protocol · channels

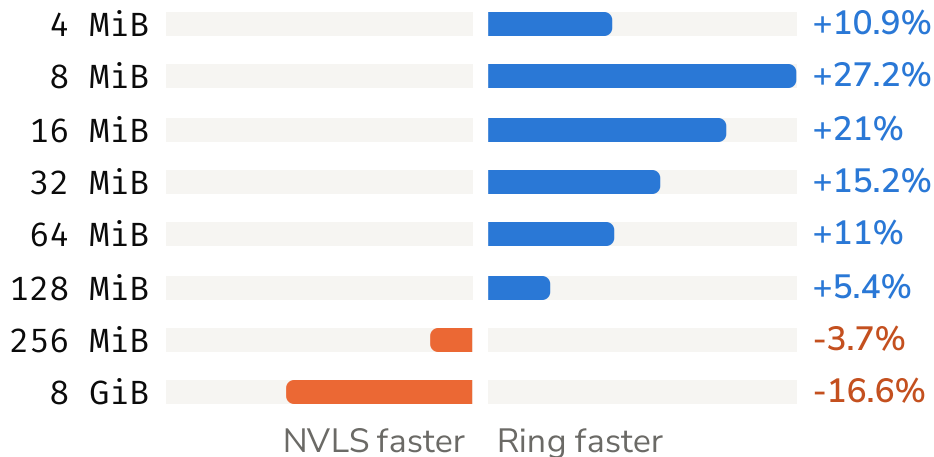
launch GPU kernels on every rank

move data over NVLink or the network

NCCL is NVIDIA's library for GPU collectives.

The default choice leaves performance on the table

8× NVIDIA B300, NVLink · AllReduce bus bandwidth, Ring vs. NCCL default (NVLS)



NCCL picks NVLS at every size: its cost model is fixed per communicator. A global `NCCL_ALGO=Ring` loses at large sizes.

One testbed is the easy case. In practice:

Run-time drift. Links degrade, stragglers appear, jobs share the network.

Mixed training. GPU generations in one job, or across datacenters.

Mixed inference. Model layers split across different GPU types.

Needed: a per-call policy, per deployment, able to use run-time measurements. Today, that means a **tuner plugin**.

Plugins let operators and vendors customize NCCL

tuner → *choose*

Per call, picks the choice by editing a cost table (estimated time per algorithm × protocol).

profiler → *launch, move*

Gets timestamped events.

net → *move*

Network transport.

env

Runtime parameters.

All four: native shared libraries (`dlopen`), one copy per rank. Each rank's tuner decides alone.

But a plugin can stop the whole job, in two ways

A bug on one rank

Native code in the process: a null dereference crashes it, a loop or race stalls it, and every other rank waits.

Seen in practice: NCCL's inspector plugin (use-after-free, deadlock); profiler plugins (segfaults on multi-GPU nodes).

Ranks choose differently

Each rank's tuner decides alone. On rank-local data, e.g. its own measured latency, ranks can pick different algorithms and **the job hangs**.

NCCL aligns inputs when the communicator is created; **the tuner API never states this, and nothing checks it**. Tuning on measurements feeds the tuner exactly such data.

We want per-call policies that cannot crash the job and cannot make ranks disagree.

NCCL's plugins look like kernel hooks, so use eBPf

Kernel	NCCL
Packet-processing hooks	net plugin
Tracepoints, policy hooks	tuner, profiler
Verifier + JIT	same, in userspace (bpftime), with a cross-rank check
Maps	same, plus a distributed map

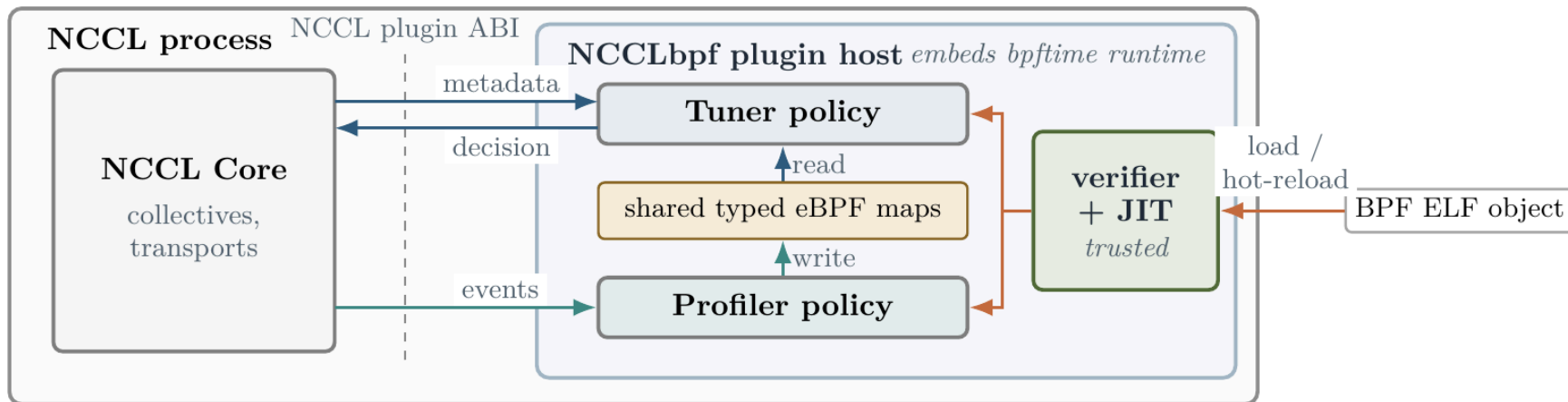
eBPf already handles plugin bugs

The verifier rejects crashing or non-terminating policies before they run.

Rank disagreement is new: the verifier cannot see it

Kernel hooks are local: one CPU, one decision. A verifier checks one program on one rank; whether ranks agree is outside its view.

NCCLbpf: verified policies behind NCCL's plugin interface



Trusted: host, JIT, verifier. The host checks outputs against run-time limits and NCCL's rules.

Checked: operators' policies, picking only among NCCL's built-in choices.

Not covered: bad decisions, resource exhaustion, side channels, trusted-part bugs.

Original Figure 1. Implementation: NCCL 2.29.7 + bpftime; ~2,500 lines of host, ~500 lines of policies; no NCCL changes.

Example: measure per rank, decide on agreed values

Paper Listing 1, with `latency_map` declared as a distributed map

```
/* profiler: write this rank's measurement */
__u32 key = 0;
struct latency_state *st =
    bpf_map_lookup_elem(&latency_map, &key);
if (!st) return 0;
st->avg_latency_ns = ctx->latency_ns;
/* tuner: read the value agreed across ranks */
__u32 key = 0;
struct latency_state *st =
    bpf_map_lookup_elem(&latency_map, &key);
if (!st) { ctx->n_channels = 4; return 0; }
/* ... algorithm by size, channels by latency ... */
```

Maps connect plugins

The profiler writes, the tuner reads the same `latency_map`. Drop a null check and the policy never loads.

Every rank makes the same choice

The profiler writes this rank's slot. The tuner reads a snapshot that is identical on every rank at each call: the latency, and whether the entry exists (`if (!st)`).

A cross-rank check: choices depend only on shared inputs

Same policy + same inputs $\Rightarrow$ same choice. At load, a taint analysis follows data and control flow from each input to each output.

What we check

Output	Agree on	May depend on
algorithm, protocol, channels	all ranks	shared inputs only: call arguments, comm profile from init, distributed maps
net plugin, per connection	both ends	+ the connection handshake
profiler; net plugin's local choices	no one	anything

In the latency-based tuner, two flows reach channels: `if (!st)` (control) and the latency (data).

Ordinary map ✗ both carry this rank's state: rejected.

Distributed map ✓ both carry the agreed snapshot: accepted.

A distributed eBPF map: measure locally, agree globally

The `latency_map`, declared as distributed

```
struct {
    __uint(type, BPF_MAP_TYPE_DIST_ARRAY);
    __type(value, struct latency_state);
    __uint(epoch_calls, 1024);          /* K */
} latency_map SEC(".maps");
/* merge: same program on every rank */
SEC("dist_merge")
int worst_rank(struct merge_ctx *m) {
    __u64 w = 0;
    for (int r = 0; r < m->n_ranks; r++)
        w = max(w, m->slot[r].avg_latency_ns);
    m->out->avg_latency_ns = w;
    return 0;
}
```

Programs unchanged: writes go to this rank's slot, lookups return the agreed value.

1 · Write locally. Each rank writes its own slot; the hot path stays one lookup.

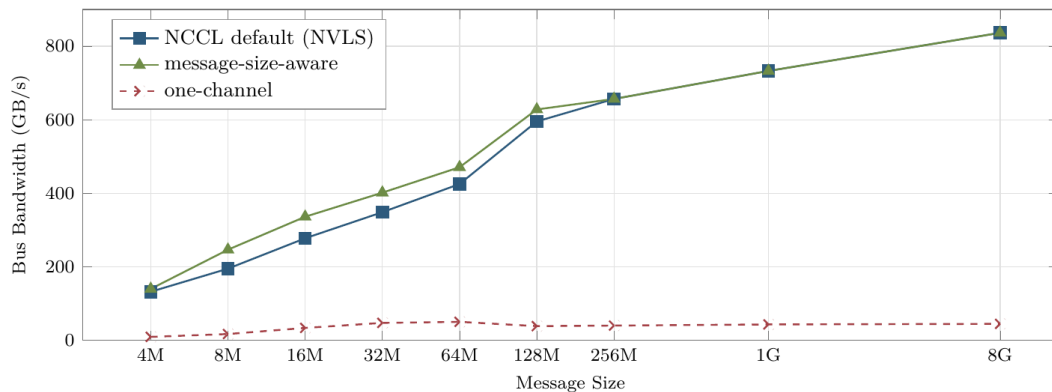
2 · Exchange every K calls. Shared memory in a node, the runtime across nodes; never through the NCCL call being tuned.

3 · Merge deterministically. Same verified merge, same slots $\Rightarrow$ same value. No leader.

4 · Activate at an agreed call. Never fall back. Reads pass the cross-rank check.

Cheap, safe, and a 20-line policy gains up to 27%

8× NVIDIA B300 with NVLink, NCCL 2.29.7.



Green: Ring at 4–192 MiB, default elsewhere. +5.5–26.5% in range, same outside.

AllReduce bus bandwidth, steady state. Original Figure 2.

80–130 ns per decision · unsafe policies (null deref, out of bounds, unbounded loop, ...) rejected at load
· hot reload in 1.07 μ s, no call lost

Verified, composable policies for NCCL

NCCLbpf

eBPF policies behind NCCL's tuner, profiler and net plugins, no NCCL changes: **verified** at load, **composable** through typed maps, **hot-reloaded** without a restart.

A cross-rank check

Outputs that must agree depend only on agreed inputs, through data and control flow; checked at load.

A distributed eBPF map

Write locally, run the same verified merge on every rank, activate at an agreed call.

Kernel hooks are local. **Distributed hooks must agree**, in NCCL, AMD's RCCL or MPI.

Code: github.com/eunomia-bpf/nccl-eBPF

Thanks to Verda for the testbed · Questions?