A Hardware Accelerator for Protocol Buffers

Sagar Karandikar [1,2], Chris Leary [2], Chris Kennelly [2], Jerry Zhao [1], Dinesh Parimi [1], Borivoje Nikolić [1], Krste Asanović [1], Parthasarathy Ranganathan [2]

[1] UC Berkeley, [2] Google

sagark@eecs.berkeley.edu

MICRO ‘21 paper: https://sagark.org/assets/pubs/protoacc-micro2021-preprint.pdf

What is Protocol Buffers (“protobuf”)?

- Google’s serialization framework
- Open-source and also widely used outside Google
  - https://github.com/protocolbuffers/protobuf
- Serialization/Deserialization are foundational operations in Warehouse Scale Computers (WSCs). Two key use cases:

01	**Inter-service communication via RPC**	• Serialize arguments to send RPC request • Deserialize return values from received RPC response • Potentially high fan-out
02	**Formatting data for storage**	• Storage APIs commonly expect data as contiguous seq. of bytes • Serialize in-mem service/app data to seq. of bytes for storage • Deserialize seq. of bytes read from storage to in-mem format usable by services

Why accelerate protobuf?

5%

of fleet-wide cycles spent in protobuf

Google, 2015 [1]

6%

of cycles in key microservices spent in serialization/deserialization

Facebook, 2020 [2]

9.6%

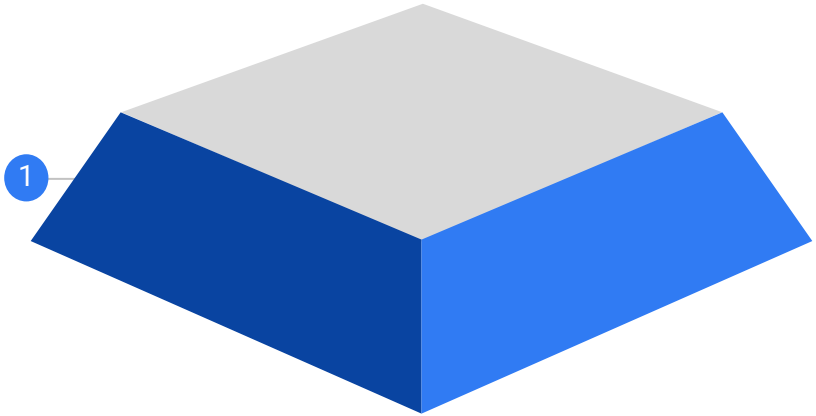
of fleet-wide cycles spent in protobuf

Google, now [this work]

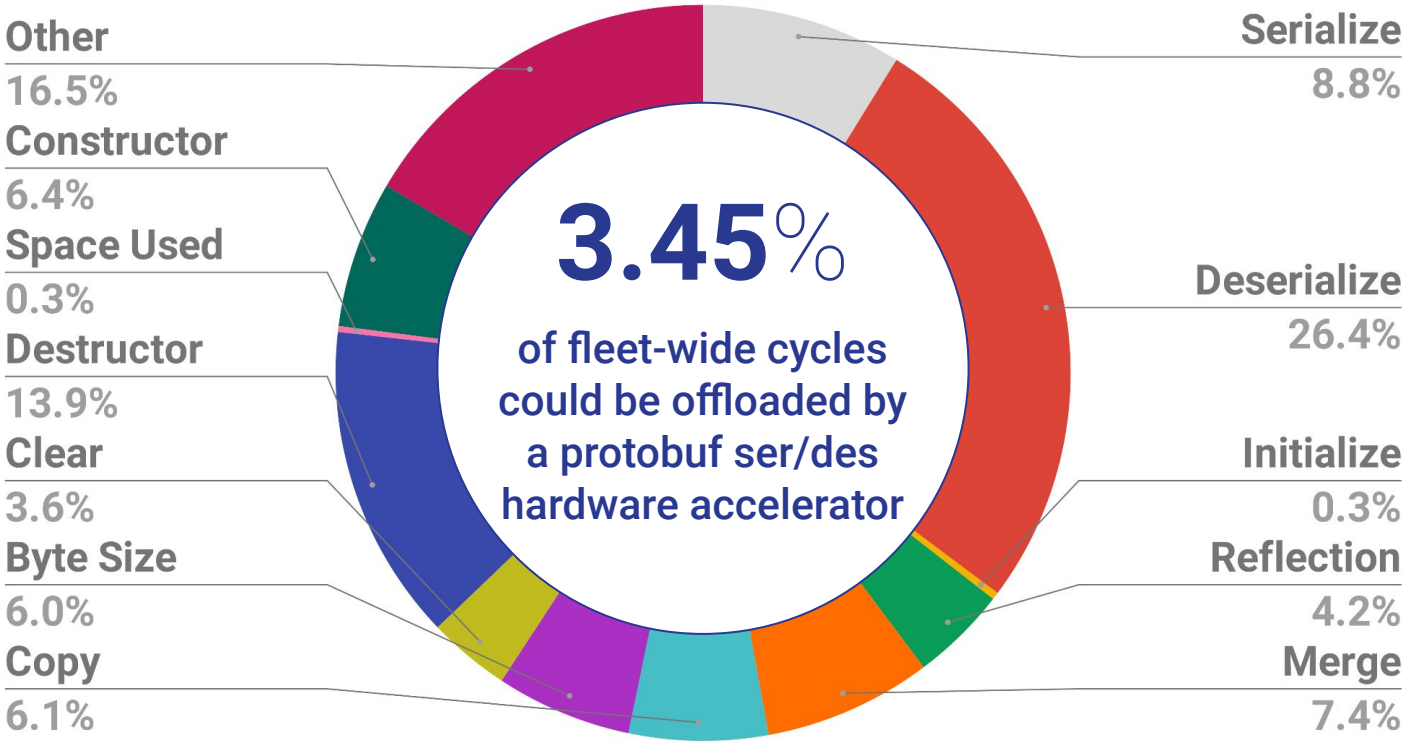
Key Contributions

An analysis of fleet-wide protobuf usage at Google

Key insights for serialization framework and serialization accelerator design, based on fleet-wide profiling at Google.



Profiling insight #1: Acceleration opportunity



Fleet-wide C++ protobuf cycles by operation.
Total pie = 8.4% of fleet-wide cycles.

(1) An analysis of fleet-wide protobuf usage at Google

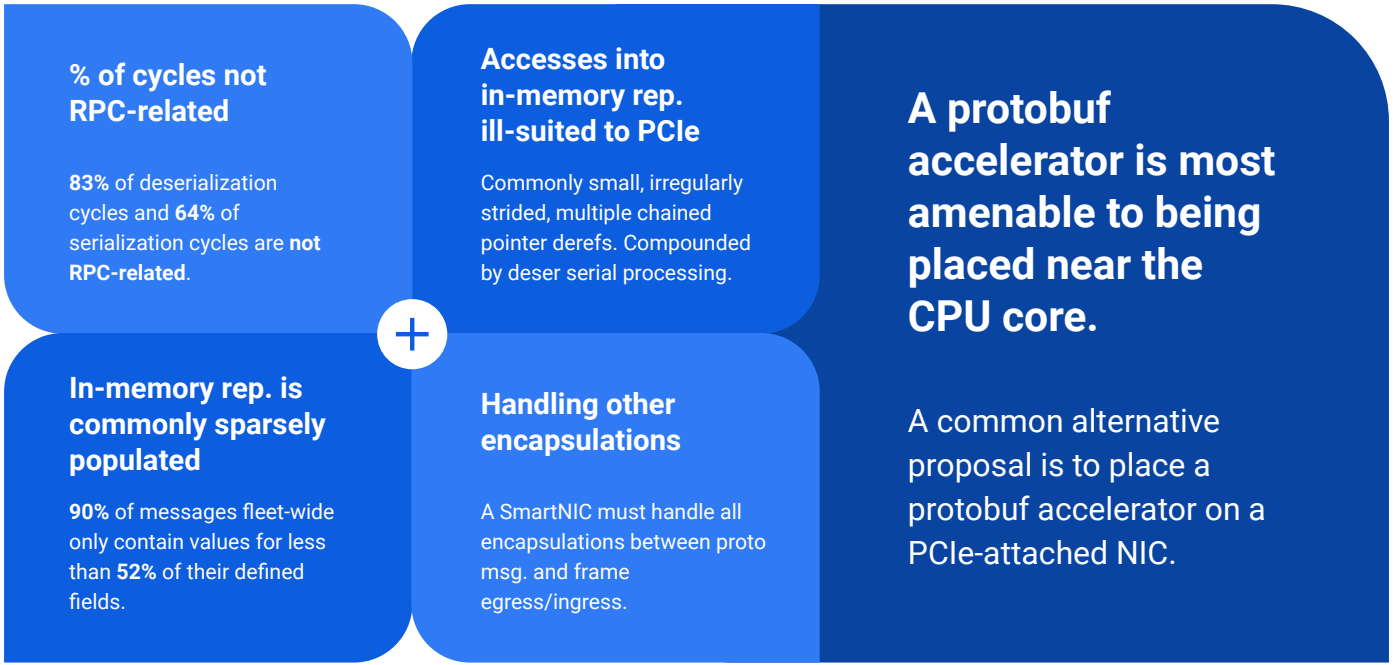
Profiling insight #2: HW feasibility

`proto3` was released in mid-2016, but 96% of protobuf bytes serialized/deserialized remain defined in `proto2`

Usage of serialization framework APIs and formats tends to be stable over time, making hardware acceleration viable.

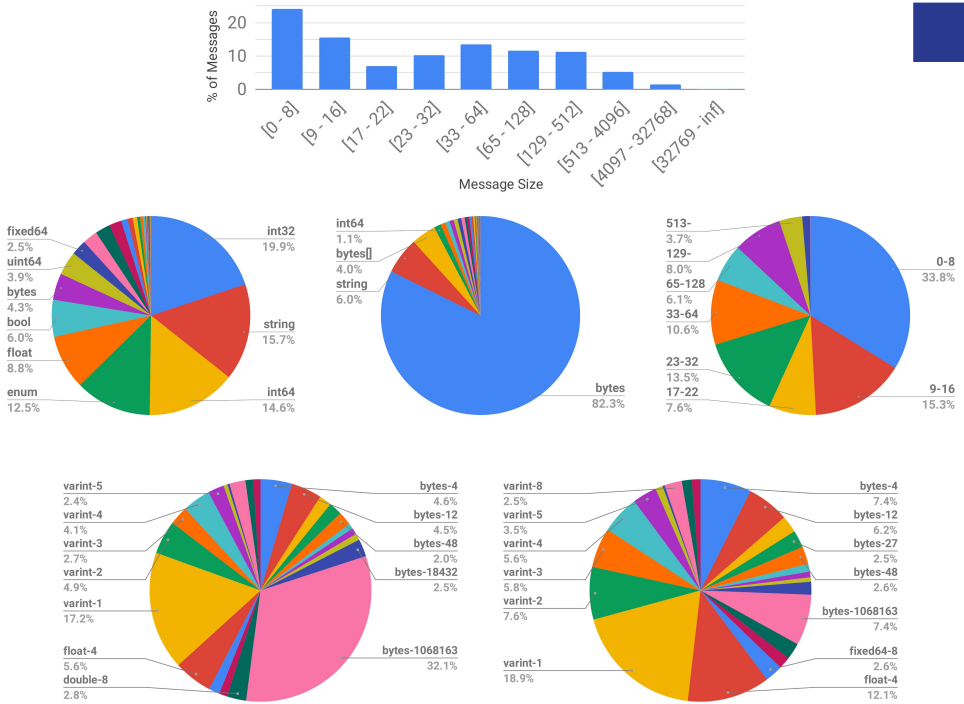
(1) An analysis of fleet-wide protobuf usage at Google

Profiling insight #3: Near-core, not PCIe-attached



(1) An analysis of fleet-wide protobuf usage at Google

Profiling insight #4: A variety of message shapes



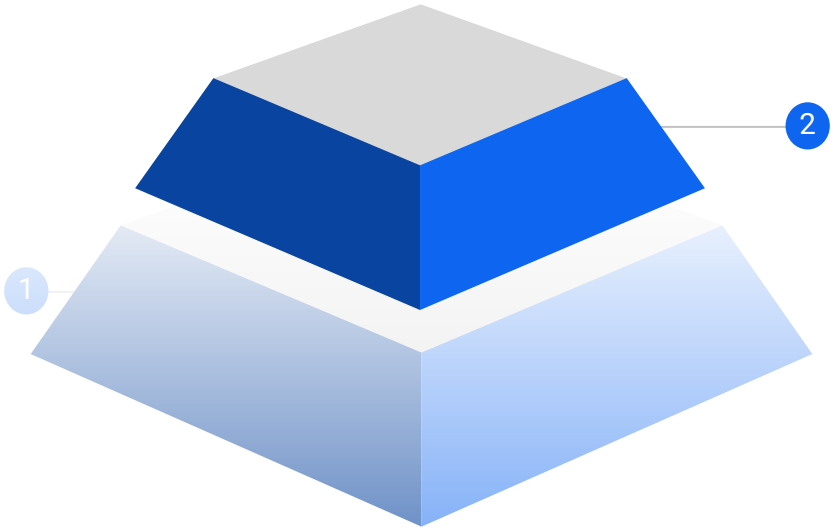
- 56% of messages are 32B or less
  - Accelerator should operate at entire msg. granularity, near-core, without CPU intervention
- Lots of data is in large byte fields
  - Accelerator needs efficient memcpy support
- Ser/des CPU cycles split across many field types
  - Accelerator needs to efficiently handle all field types
- 99.999% of bytes of protobuf data are at sub-message depth 25 or less

(1) An analysis of fleet-wide protobuf usage at Google

Key Contributions

An analysis of fleet-wide protobuf usage at Google

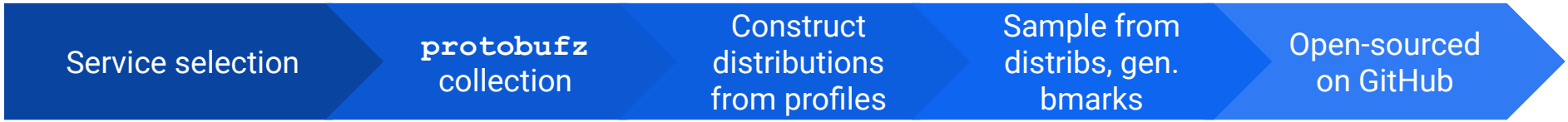
Key insights for serialization framework and serialization accelerator design, based on fleet-wide profiling at Google.



An open-source hyperscale protobuf benchmark

HyperProtoBench, an open-source benchmark suite representative of key protobuf users at Google.
github.com/google/HyperProtoBench

HyperProtoBench: Open-source protobuf benchmarks representative of key protobuf-user services at Google



Select the top five fleet-wide serialization and deserialization users. Four services appear in both = six benchmarks.

Collect fleet-wide protobuf message “shape” information for each service.

Construct protobuf message shape distributions based on the per-service profiling data.

Per-selected service, generate a `.proto` file with message definitions and a C++ benchmark that constructs, mutates, and ser/des messages.

github.com/google/HyperProtoBench

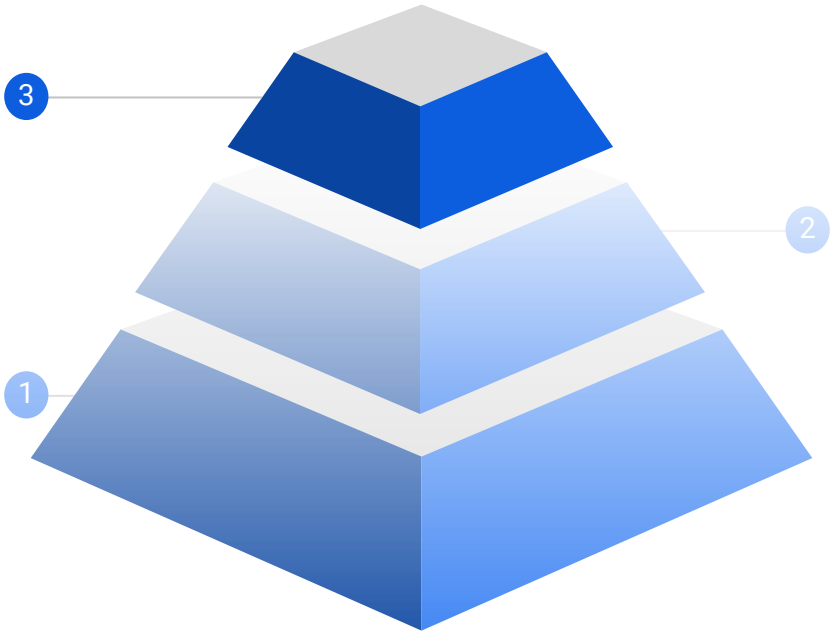
Key Contributions

An open-source RTL protobuf accelerator

A novel, open-source protobuf accelerator design aligned with profiling insights, implemented in RTL and integrated into a RISC-V SoC w/BOOM OoO core.
github.com/ucb-bar/protoacc

An analysis of fleet-wide protobuf usage at Google

Key insights for serialization framework and serialization accelerator design, based on fleet-wide profiling at Google.



An open-source hyperscale protobuf benchmark

HyperProtoBench, an open-source benchmark suite representative of key protobuf users at Google.
github.com/google/HyperProtoBench

Generating “programming” for a protobuf accelerator

- Accelerator Descriptor Tables (ADTs)
  - Describe the layout of messages in application memory
  - Automatically generated by modified `protoc` compiler
  - Created/populated ***once*** at application load time
  - One per message-***type***
- `hasbits` bit field
  - HW serializer must know which fields are present (i.e., set) in the in-memory C++ representation
  - Standard protobuf C++ message objects **already track presence via `hasbits`**
    - A set of bits where the bit corresponding to a field is set if the field is present
  - Modify `protoc` to emit sparse `hasbits` encoding for efficient accelerator access
  - One per message-***instance***

*In contrast, the closest prior work [3] relies on per message-**instance** “schemas” maintained by code added to all field setters/clear methods, adding significant CPU/memory overhead.*

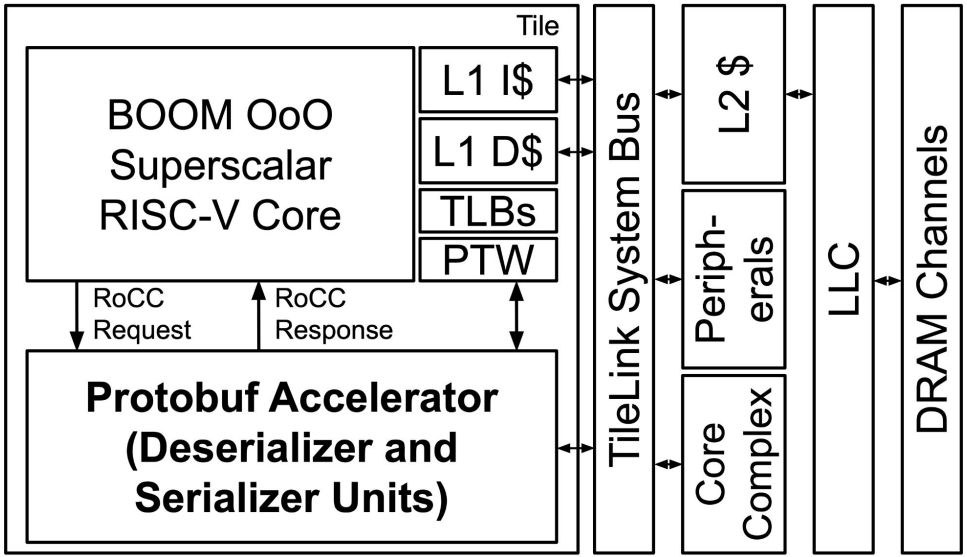
Profiling insight #5: Protobuf accelerator programming tradeoffs

Our software modifications for accelerator support are more efficient in CPU/memory overhead terms than prior work [3] for over 92% of fleet-wide messages.

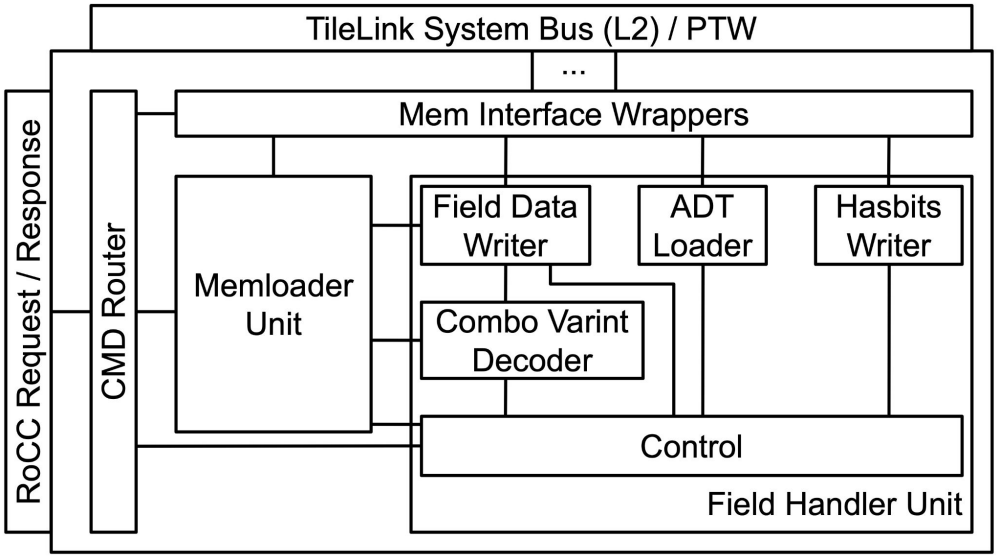
(3) An open-source RTL protobuf accelerator

Protobuf accelerator: System overview

- Accelerator written in Chisel RTL [4]
- Integrated into Chipyard RISC-V SoC generator [5]
- SonicBOOM OoO RISC-V as application core [6]
  - IPC-comparable on SPEC17 with ARM Cortex A72-like cores
- Accelerator interfaces:
  - Custom RISC-V instructions from core via RoCC
  - Coherent memory access (via L2) using 128-bit TileLink
  - Page-table walker access (accelerator operates on virtual addresses)



Protobuf accelerator: Deserializer overview



Low-level SW API:

- One RoCC setup instruction to supply accelerator arena (amortized across several messages)
- Two non-blocking RoCC instructions to kick-off deserialization
- One RoCC instruction to block on all in-flight deserializations

Key features/insights:

- Combinational varint decode, sub-message support (on-chip metadata stack sized to 25 based on profiling)
- Automatically allocates/constructs C++ objects for sub-messages, `std::string` for strings (including SSO), repeated field objects in accel. arena
- Also populates `hasbits` required for serialization-side

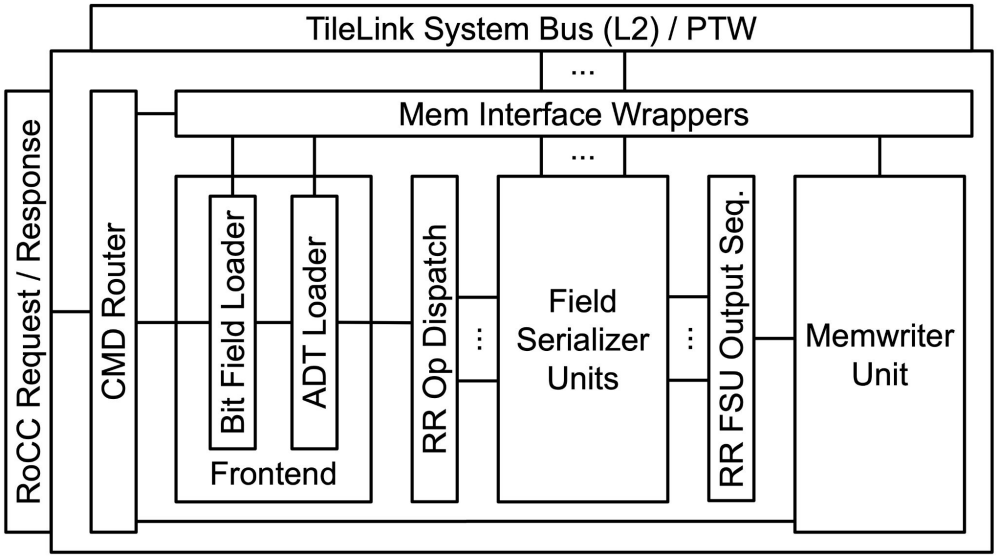
Protobuf accelerator: Serializer overview

Low-level SW API:

- One RoCC setup instruction to supply accelerator arena (amortized across several messages)
- Two non-blocking RoCC instructions to kick-off serialization
- One RoCC instruction to block on all in-flight serializations

Key features/insights:

- Combinational varint encode, sub-message support (on-chip metadata stack sized to 25 based on profiling)
- Fields are handled in reverse field # order and data is written from high-to-low address: more efficient handling of length-delimited lengths
- Handling of individual fields parallelized across multiple field serializer units



Key Contributions

An open-source RTL protobuf accelerator

A novel, open-source protobuf accelerator design aligned with profiling insights, implemented in RTL and integrated into a RISC-V SoC w/BOOM OoO core.
github.com/ucb-bar/protoacc

An analysis of fleet-wide protobuf usage at Google

Key insights for serialization framework and serialization accelerator design, based on fleet-wide profiling at Google.

A reproducible, end-to-end evaluation

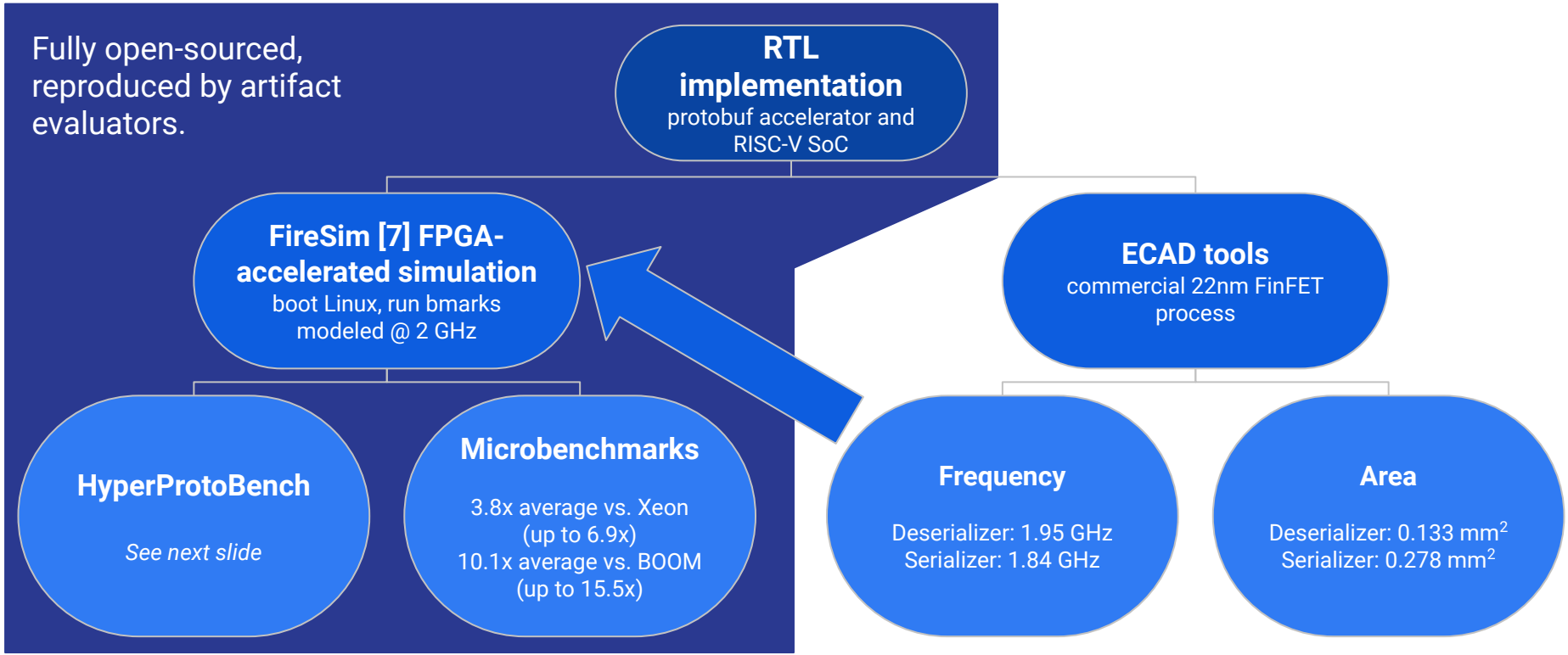
HyperProtoBench and μbmarks running on RTL-impl. of accelerated system, cycle-exactly simulated in FireSim. Up to 6.9x improvement vs. Xeon, 15.5x vs. BOOM. Reproduced by artifact evaluators.

An open-source hyperscale protobuf benchmark

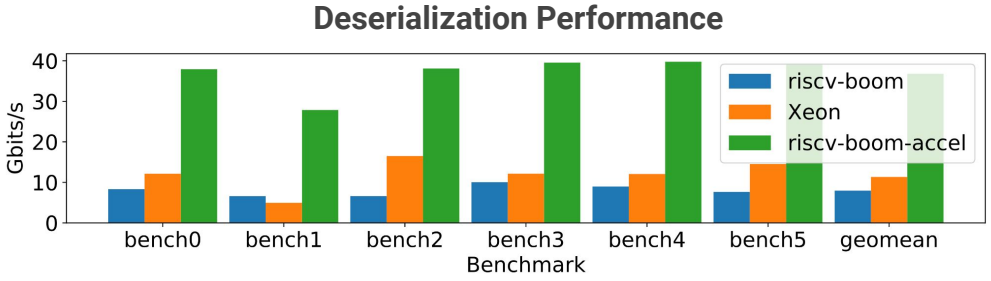
HyperProtoBench, an open-source benchmark suite representative of key protobuf users at Google.
github.com/google/HyperProtoBench

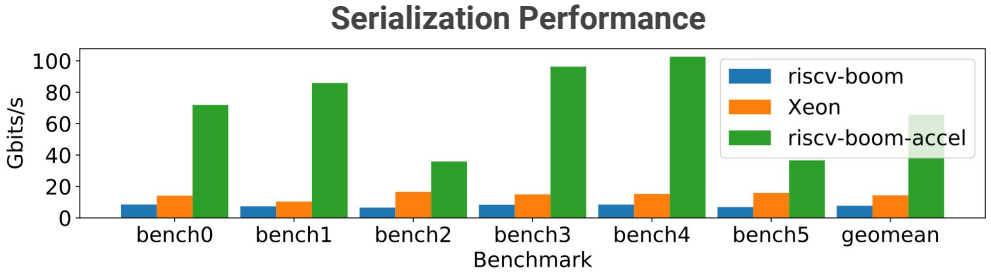
MICRO-54 Distinguished Artifact Winner!

Evaluation methodology/overview



HyperProtoBench results





A savings of 2.5% of fleet-wide cycles

Accelerator achieves geomean

6.2x improvement
vs. SonicBOOM

3.8x improvement
vs. Xeon E5-2686 v4 *

* despite RISC-V SoC's weaker uncore/supporting components

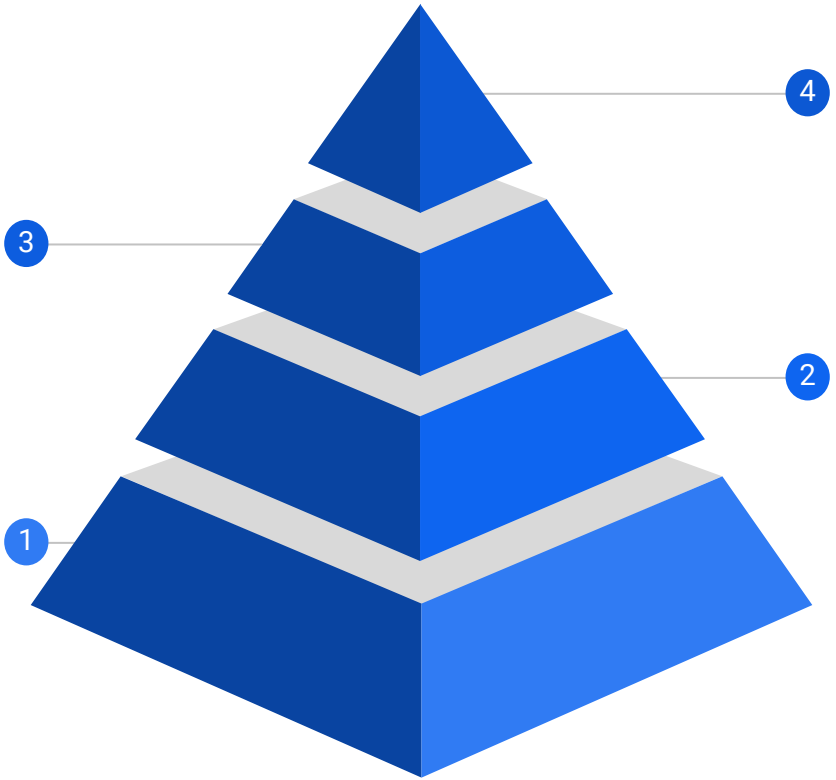
Conclusion

An open-source RTL protobuf accelerator

A novel, open-source protobuf accelerator design aligned with profiling insights, implemented in RTL and integrated into a RISC-V SoC w/BOOM OoO core.
github.com/ucb-bar/protoacc

An analysis of fleet-wide protobuf usage at Google

Key insights for serialization framework and serialization accelerator design, based on fleet-wide profiling at Google.



A reproducible, end-to-end evaluation

HyperProtoBench and μbmarks running on RTL-impl. of accelerated system, cycle-exactly simulated in FireSim. Up to 6.9x improvement vs. Xeon, 15.5x vs. BOOM. Reproduced by artifact evaluators.

An open-source hyperscale protobuf benchmark

HyperProtoBench, an open-source benchmark suite representative of key protobuf users at Google.
github.com/google/HyperProtoBench

References

[1] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. **Profiling a warehouse-scale computer.** In Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA '15). Association for Computing Machinery, New York, NY, USA, 158–169. DOI:https://doi.org/10.1145/2749469.2750392

[2] Akshitha Sriraman and Abhishek Dhanotia. 2020. **Accelerometer: Understanding Acceleration Opportunities for Data Center Overheads at Hyperscale.** In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20). Association for Computing Machinery, New York, NY, USA, 733–750. DOI:https://doi.org/10.1145/3373376.3378450

[3] Arash Pourhabibi, Siddharth Gupta, Hussein Kassir, Mark Sutherland, Zilu Tian, Mario Paulo Drumond, Babak Falsafi, and Christoph Koch. 2020. **Optimus Prime: Accelerating Data Transformation in Servers.** In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20). Association for Computing Machinery, New York, NY, USA, 1203–1216. DOI:https://doi.org/10.1145/3373376.3378501

[4] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. **Chisel: constructing hardware in a Scala embedded language.** In Proceedings of the 49th Annual Design Automation Conference (DAC '12). Association for Computing Machinery, New York, NY, USA, 1216–1225. DOI:https://doi.org/10.1145/2228360.2228584

[5] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Yakun Sophia Shao, Krste Asanovic, and Borivoje Nikolic. 2020. **Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs.** In IEEE Micro, vol. 40, no. 4, pp. 10-21, 1 July-Aug. 2020, DOI: 10.1109/MM.2020.2996616.

[6] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. 2020. **SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine.** In Fourth Workshop on Computer Architecture Research with RISC-V (CARRV 2020). https://carrv.github.io/2020/papers/CARRV2020_paper_15_Zhao.pdf

[7] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Qijing Huang, Kyle Kovacs, Borivoje Nikolic, Randy Katz, Jonathan Bachrach, and Krste Asanović. 2018. **FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud.** In Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA '18). IEEE Press, 29–42. DOI:https://doi.org/10.1109/ISCA.2018.00014

A Hardware Accelerator for Protocol Buffers

Sagar Karandikar [1,2], Chris Leary [2], Chris Kennelly [2], Jerry Zhao [1], Dinesh Parimi [1], Borivoje Nikolić [1], Krste Asanović [1], Parthasarathy Ranganathan [2]

[1] UC Berkeley, [2] Google

sagark@eecs.berkeley.edu

MICRO ‘21 paper: https://sagark.org/assets/pubs/protoacc-micro2021-preprint.pdf