Tags: A Framework for Distributed Event Ordering

Paul Mure
Stanford University
USA

Nathan Zhang
Stanford University
USA

Caroline Trippel
Stanford University
USA

Kunle Olukotun
Stanford University
USA

Abstract

The rise of large-scale machine learning applications necessitates throughput-oriented machines. Dataflow computing has thus regained traction due to its ability to capitalize on fine-grained parallelism in both the vector and pipeline dimensions. Modern reconfigurable dataflow accelerators (RDAs) in particular support expressing highly customizable memory orderings, enabling algorithm developers to explore specialized synchronization schemes tailored to each application. Unfortunately, a generalized framework for reasoning about memory orderings that accounts for the unique constraints of RDAs, like strict pipeline timing, is lacking. Such a framework can provide stronger correctness guarantees by making the programming model more amenable to formal verification. In this talk, we will describe the shortcomings of analogous CPU-based frameworks and propose a new approach for RDAs, called Tags, based on a generalized notion of the token-credit system. We also outline potential research directions towards integrating Tags into new languages and compilers and exploring relationships between Tags and session types.

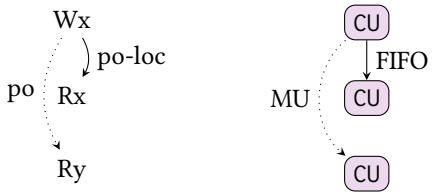
1 Motivation

The recent revival of dataflow computing in the form of reconfigurable dataflow accelerators (RDAs) [14], and their growing proliferation [13], poses a new set of challenges for programming language and compiler designers [21]. Specifically, we call to attention the importance of reasoning about event ordering, namely events related to memory operations. Current programming models for RDAs lack a formal foundation, and thus a principled framework, for rigorously reasoning about memory ordering. Traditional CPU-centric programming models define memory ordering behaviors by a hardware memory consistency model (i.e. x86-TSO, ARM, PowerPC) and a language memory model (C++, Java, Go). Thus, existing generalized memory modeling frameworks assume a processor-centric view of shared-memory [1–3, 11, 15, 17, 19]. However, RDAs exhibit several important distinctions: hundreds of incoherent memory scopes which feature *explicit* memory ordering semantics, hundreds of PEs which *locally* enforce program order for a handful of memory operations, pairwise FIFOs which enforce *implicit* memory ordering semantics, and a desire to sometimes bound reordering of memory accesses rather than disallow them entirely. Unfortunately, due to the discrepancies between RDAs and processor-like architectures, existing memory modeling frameworks do not provide the desired set of properties for RDAs, summarized below.

Fine-grained Dataflow. RDAs are characterized by a sea of simple software-reconfigurable compute units (CUs) and memory units (MUs) connected by an on-chip network. In processor-based architectures, there is a clear separation of ordering concerns between hardware and software. This is not true in RDAs, where the task of ensuring application correctness is almost exclusively offloaded to software. For example, as shown in fig. 1a, CPU-based memory models are concerned with sequences of instructions in program order and when their effects to memory are visible to each other. Most CPUs today have hardware-defined coherence protocols that guarantee same-address accesses follow program order, while different-address access often require additional software-defined synchronization. For a typical RDA, each CU executes its configured operations sequentially, but between multiple CUs, the hardware can only guarantee FIFO ordering in per-link buffers between each pair of CUs. Larger memories that do not fit in these buffers must go through an MU, at which point any orderings guarantees must be enforced by software. This can be seen as an extreme version of scoped memory with hundreds of small reconfigurable regions that can be independently configured. For a compiler to effectively maximize performance on such a machine, a flexible framework is needed to allow users to easily communicate minimal ordering requirements that maximize performance.

Bounded Desynchronization. While many programs have a strict notion of correctness, stochastic and iterative algorithms such as stochastic gradient descent, matrix completion, and graph cuts have far looser requirements [16]. Instead, these applications only require that reads and writes

PLARCH '23, June 17, 2023, Orlando, FL



(a) CPU **(b)** RDA

Figure 1. Differences between CPU memory model and RDA memory model. A CPU-based model shown in fig. 1a reasons about when the effects of instructions are visible by other instructions. Typically, the hardware will preserve same-address program order (po-loc), whereas different-address program order (po) is not always guaranteed. For RDAs shown in fig. 1b, hardware typically provides small buffers between CUs that follow hardware-defined FIFO-ordering, whereas larger intermediate data must be communicated through an MU with software-defined ordering.

execute mostly in program order for the application-level result to eventually converge. Existing memory models do not provide a way of bounding the staleness of accesses, and rely entirely on ad hoc software that is inflexible to changes. We will expound further on this point in section 2.2.

2 Envisioned Approach: Tags

```
Tag (id: Nat):
  outlet:
    publish : OutState → SignalType
  inlet:
    ready: SignalType → InState → Bool
    update: SignalType → InState → Bool → InState
```

We propose Tags, a general programming framework for specifying distributed event ordering based on distributed *local state* and targeted *communications*. The design of Tags is inspired by early work on Synchronous Dataflow [10], which models dataflow computation as a set of actors communicating through tokens to signal inter-node dependencies. Tags can be seen as a generalization of this token system.

2.1 Specification

A Tag is a user-specified dependency between multiple distributed operations, each with local state invisible to the others. Each Tag is uniquely identified with an integer `id`, where `outlets` and `inlets` under the same Tag `id` communicate with each other. For a unique Tag `id`, a node can either contain an `outlet` or an `inlet`. An `outlet` represents a producer of some kind, where the `publish` function is set to trigger whenever the body of the node finishes an iteration of computation. The `outlet` function views some prescribed state of the producer node, with polymorphic type Φ, signals an update of polymorphic type Δ over the "wire". Nodes with associated `inlets` evaluate the `ready` function upon receiving such an update signal from an outlet with the same `id`. The `ready` function uses the update signal received of type Δ, in combination with some polymorphic local state of type Ψ, to determine whether the associated actions of the node are ready to be scheduled, denoted with type **Bool**. Lastly, the update signal, the consumer node's local state, and the current readiness trigger an update to its local state in the `update` function.

2.2 Discussion: Bounded Desynchronization

One application of Tags is to easily describe the notion of *bounded desynchronization*: when an algorithm might tolerate slight incoherence to reduce synchronization overheads within a reasonable limit. Prior works like HOGWILD![16] have analyzed the benefits of asynchronous stochastic gradient descent (SGD) without locks. While HOGWILD! achieves speedups by decreasing the execution time of each iteration through reduced communication overhead, analysis has found that the convergence rate suffers as data staleness increases [5, 16]. Tags offer to balance the staleness-synchronization trade-off by allowing the user to easily describe a bound on data staleness by communicating current iteration counts between the nodes. One potential scheme for staleness-aware asynchronous SGD is described by Zhang et al. [20] which leverages MPI to achieve the desired effects. However, MPI is a processor-centric model that is more appropriate for data centers than RDAs.

3 Future Work

We are currently exploring these three directions:

Constraints. To satisfy hardware plausibility and prevent catastrophic failures, there must be a well-founded set of constraints on what the prescribed functions can do. For example, `ready` is monotonically non-decreasing with respect to the `InState`. This means that if a previous call to `ready` returns true, "increasing" the `InState` should not result in a false enable signal, which would lead to unprocessed elements and potential deadlock.

Language Design. Tags can potentially play a crucial role in designing verifiable high-performance HLS languages for RDA. To this end, we are currently working towards integrating Tags into a formally verified HLS language for programming RDAs based on an interactive proof assistant.

Correspondence to Session Types. Session types are promising directions in formally reasoning about concurrent programs, but a lower-level IR is needed to realize them on hardware [6, 7, 18]. A potential correspondence exists between Tags and session types which we are currently working to concretize. Such a correspondence, once established, can enable the wealth of knowledge accrued in session types to transfer to a system that is more friendly to low-level implementation [4, 8, 9, 12].

References

[1] Jade Alglave, Will Deacon, Richard Grisenthwaite, Antoine Hacquard, and Luc Maranget. 2021. Armed Cats: Formal Concurrency Modelling at Arm. *ACM Trans. Program. Lang. Syst.* 43, 2, Article 8 (jul 2021), 54 pages. https://doi.org/10.1145/3458926

[2] Jade Alglave, Luc Maranget, and Michael Tautschnig. 2014. Herding Cats: Modelling, Simulation, Testing, and Data Mining for Weak Memory. *ACM Trans. Program. Lang. Syst.* 36, 2, Article 7 (jul 2014), 74 pages. https://doi.org/10.1145/2627752

[3] Jennifer Brana, Brian C. Schwedock, Yatin A. Manerkar, and Nathan Beckmann. 2023. Kobold: Simplified Cache Coherence for Cache-Attached Accelerators. *IEEE Computer Architecture Letters* (2023), 1–4. https://doi.org/10.1109/LCA.2023.3269399

[4] Zak Cutner, Nobuko Yoshida, and Martin Vassor. 2022. Deadlock-Free Asynchronous Message Reordering in Rust with Multiparty Session Types. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Seoul, Republic of Korea) *(PPoPP '22)*. Association for Computing Machinery, New York, NY, USA, 246–261. https://doi.org/10.1145/3503221.3508404

[5] Christopher M De Sa, Ce Zhang, Kunle Olukotun, Christopher Ré, and Christopher Ré. 2015. Taming the Wild: A Unified Analysis of Hogwild-Style Algorithms. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2015/file/98986c005e5def2da341b4e0627d4712-Paper.pdf

[6] Kohei Honda. 1993. Types for dyadic interaction. In *CONCUR'93*, Eike Best (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 509–523.

[7] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. 1998. Language primitives and type discipline for structured communication-based programming. In *Programming Languages and Systems*, Chris Hankin (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 122–138.

[8] Jules Jacobs, Stephanie Balzer, and Robbert Krebbers. 2022. Multiparty GV: Functional Multiparty Session Types with Certified Deadlock Freedom. *Proc. ACM Program. Lang.* 6, ICFP, Article 107 (aug 2022), 30 pages. https://doi.org/10.1145/3547638

[9] Wen Kokke and Ornela Dardha. 2021. Deadlock-Free Session Types in Linear Haskell. In *Proceedings of the 14th ACM SIGPLAN International Symposium on Haskell* (Virtual, Republic of Korea) *(Haskell 2021)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3471874.3472979

[10] E.A. Lee and D.G. Messerschmitt. 1987. Synchronous data flow. *Proc. IEEE* 75, 9 (1987), 1235–1245. https://doi.org/10.1109/PROC.1987.13876

[11] Daniel Lustig, Sameer Sahasrabuddhe, and Olivier Giroux. 2019. A Formal Analysis of the NVIDIA PTX Memory Consistency Model. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 257–270. https://doi.org/10.1145/3297858.3304043

[12] Nicholas Ng and Nobuko Yoshida. 2016. Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis. In *Proceedings of the 25th International Conference on Compiler Construction* (Barcelona, Spain) *(CC 2016)*. Association for Computing Machinery, New York, NY, USA, 174–184. https://doi.org/10.1145/2892208.2892232

[13] Raghu Prabhakar, Sumti Jairath, and Jinuk Luke Shin. 2022. SambaNova SN10 RDU: A 7nm Dataflow Architecture to Accelerate Software 2.0. In *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, Vol. 65. 350–352. https://doi.org/10.1109/ISSCC42614.2022.9731612

[14] Raghu Prabhakar, Yaqi Zhang, David Koeplinger, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. 2017. Plasticine: A Reconfigurable Architecture For Parallel Paterns. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON, Canada) *(ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 389–402. https://doi.org/10.1145/3079856.3080256

[15] Christopher Pulte, Shaked Flur, Will Deacon, Jon French, Susmit Sarkar, and Peter Sewell. 2017. Simplifying ARM Concurrency: Multicopy-Atomic Axiomatic and Operational Models for ARMv8. *Proc. ACM Program. Lang.* 2, POPL, Article 19 (dec 2017), 29 pages. https://doi.org/10.1145/3158107

[16] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger (Eds.), Vol. 24. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2011/file/218a0aefd1d1a4be65601cc6ddc1520e-Paper.pdf

[17] Daniel Sorin, Mark Hill, and David Wood. 2011. *A Primer on Memory Consistency and Cache Coherence*. Morgan and Claypool Publishers.

[18] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. 1994. An Interaction-Based Language and Its Typing System. In *Proceedings of the 6th International PARLE Conference on Parallel Architectures and Languages Europe (PARLE '94)*. Springer-Verlag, Berlin, Heidelberg, 398–413.

[19] Caroline Trippel, Yatin A. Manerkar, Daniel Lustig, Michael Pellauer, and Margaret Martonosi. 2017. TriCheck: Memory Model Verification at the Trisection of Software, Hardware, and ISA. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 119–133. https://doi.org/10.1145/3037697.3037719

[20] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. 2016. Staleness-Aware Async-SGD for Distributed Deep Learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (New York, New York, USA) *(IJCAI'16)*. AAAI Press, 2350–2356.

[21] Yaqi Zhang, Nathan Zhang, Tian Zhao, Matt Vilim, Muhammad Shahbaz, and Kunle Olukotun. 2021. SARA: Scaling a Reconfigurable Dataflow Accelerator. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 1041–1054. https://doi.org/10.1109/ISCA52012.2021.00085