Streaming Tensor Programs: A Programming Abstraction for Streaming Dataflow Accelerators

Gina Sohn
Stanford University
Stanford, CA, USA
ginasohn@stanford.edu

Christophe Gyurgyik
Stanford University
Stanford, CA, USA
cpg@stanford.edu

Genghan Zhang
Stanford University
Stanford, CA, USA
zgh23@stanford.edu

Suguna Velury
Stanford University
Stanford, CA, USA
sugunav@stanford.edu

Paul Mure
MIT CSAIL
Cambridge, MA, USA
paulmure@mit.edu

Nathan Zhang
Stanford University
Stanford, CA, USA
stanfurd@stanford.edu

Kunle Olukotun
Stanford University
Stanford, CA, USA
kunle@stanford.edu

Abstract

The rise of Large Language Models necessitates hardware that can efficiently support these compute- and memory-intensive models. Streaming dataflow accelerators have gained traction as alternatives to CPUs and GPUs due to their high degree of parallelism and the ability to fuse kernels. However, programming such accelerators with existing programming models makes it difficult to fully exploit their performance benefits due to their different execution models from CPUs and GPUs. Furthermore, prior work on programming models for dataflow systems cannot fully express dynamism, such as dynamic computation graphs and dynamic tensor shapes. In this work, we describe the limitations of existing programming models for dataflow systems and propose a new programming abstraction, the Streaming Tensor Program (STeP). We also discuss plans for evaluating the expressiveness of STeP and future research direction on mapping STeP to streaming dataflow accelerators.

1 Introduction / Motivation

The advent of compute- and memory-intensive applications such as Large Language Models (LLMs) has increased the demand for high throughput accelerators, which in turn has driven research in streaming dataflow accelerators such as Reconfigurable Dataflow Accelerators (RDAs) [2, 8, 10] and Coarse-Grained Reconfigurable Architectures (CGRAs) [1, 3, 7]. These accelerators adopt a different execution model from the kernel-by-kernel execution model used in CPUs and GPUs. In the kernel-by-kernel execution model, the program gets executed by sequentially loading the inputs to memory, running a set of operations, and storing the outputs back to memory. This provides a clear boundary between kernels, and data is transferred between kernels by storing to and loading from memory. Therefore, it is natural for the programming model to take a memory-centric approach, describing each operation by indexing the inputs and outputs. On the other hand, in the dataflow execution model used in streaming dataflow accelerators, the program gets executed by spatially mapping operations to the hardware execution units and pipelining the execution between operations. This makes the spatially mapped units active most of the time with no explicit notion of instruction termination. Furthermore, when transferring data between kernels, data does not necessarily go through memory. Instead, data gets directly streamed to the hardware unit executing the next kernel. Such an execution model enables a high degree of parallelism through vector- and pipeline-parallelism and reduces the number of accesses to memory by fusing the operations. Unfortunately, programming streaming dataflow accelerators with the abstractions used for kernel-by-kernel execution models makes it difficult to exploit these benefits. Therefore, to effectively unlock the performance benefits, dataflow systems require a new stream-centric programming paradigm that maps naturally to the accelerators' execution model.

1.1 Related Work

Parallel Patterns Parallel Patterns [9] is a programming model used to program the Plasticine [10] RDA. By extending traditional functional programming, Parallel Patterns enable simple and automatic parallelization while providing higher-level abstractions instead of hardware description languages. However, Parallel Patterns still takes a memory-centric design, where explicit indices of the inputs and outputs are used to specify operations. This implicitly assumes that the inputs and outputs are materialized, so the compiler [15] for Plasticine must conduct optimizations to detect unnecessary materialization and recover the fusion capability.

Op	Type Signature & Explanation
Map	**(A→B)→St<A,α>→St<B,α>**
<α,A,B>	Apply a function on a stream element-wise.
Accum	**(()→B)→(A→B→B)→St<A,α>→St<B,$\alpha-\beta$>**
<α, β,A,B>	Accumulate St<A,α>'s lower β dimensions into a value of type B. The accumulator is initialized after accumulating every β dimension.
Flatmap	**(A→St<B,β>)→St<A,α>→St<B,$\alpha+\beta-1$>**
<α, β,A,B>	Apply an element-wise function on St<A,α>. Outputs will be flattened into a single stream.
Repeat	**St<A,α>→St<B,$\alpha+1$>→St<A,$\alpha+1$>**
<α,A,B>	Repeat each element according to the reference stream St<B,α+1>.
Partition	**St<A,α>→St<B,$\alpha-\beta$>→[St<A,α>]$_N$**
<α, β,N,A,B>	Shard St<A,α> every β dimension into N streams based on the selection in St<B,a-β>.
Reassemble	**[St<A,α>]$_N$ →St<B,β>→St<A,$\alpha+1$>**
<α, β, γ,N,A,B>	Reassemble the results from N different streams based on the selection in St<B,β>. The results are interleaved every γ dimensions.

Table 1. The type signature and description of operations. The type signature is specified in a curried format.

StreamIt StreamIt [14] is a language and compiler for stream programs such as video, digital signal processing, and networking. It adopts a stream-centric design where it defines stream types and filters that operate on streams. It does support a certain degree of dynamism, such as dynamic input and output rates. However, unlike STeP, it cannot express computation graphs that have dynamic branching behaviors because StreamIt can only statically shard and reassemble data on streams.

1.2 Key Contribution

We propose Streaming Tensor Program (STeP), a programming abstraction for streaming dataflow accelerators, which has the following contributions: (1) STeP proposes a **stream-centric programming paradigm** where programs are described as operations on streams while retaining the expressiveness to implement state-of-the-art machine learning models. (2) STeP's dynamic operations, coupled with the stream representation, can **express dynamism** such as dynamic computation graphs and dynamic tensor shapes.

2 System Overview

2.1 Types, Operations, and Functions

Stream Type: A *Stream* is an infinite sequence of values delimited by stop tokens to indicate tensor rank (or order); this is inspired by the representation used in prior work, such as SAM [6] and Revet [12], and provides the flexibility to handle dynamic tensor shapes and sparse data easily. More concretely, a stream type St<T,α> denotes a stream of values of type T with rank α where the type T can be either a *Element*, *Buffer*, or tuple of *Element*s or *Buffer*s. Since a stream is an infinite sequence, we do not allow rank-0 streams. This is defined as:

$$\texttt{St<T,}\alpha\texttt{>} = \texttt{RawSt<T,}\alpha\texttt{>}\ S_\alpha\ (\alpha\texttt{>0})$$

$$\texttt{RawSt<T,0>} = \texttt{T*}$$

$$\texttt{RawSt<T,}\alpha\texttt{>} = \texttt{(RawSt<T,}\alpha\texttt{-1>}\ S_{\alpha-1})\texttt{*}\ \texttt{RawSt<T,}\alpha\texttt{-1>}\ (\alpha\texttt{>0})$$

Element Type: This includes primitive data types such as floating point, integer, unsigned, etc. Tuples of primitive data types are also *Element* types.

Buffer Type: Since STeP is a stream-centric abstraction, we assume operations only can view inputs in a streaming (FIFO) order. However, there are occasions where the operation needs random access within certain dimensions of a stream. Therefore, we introduce a *Buffer* type, which contains *Element*s or tuples of *Element*s. For example, St<Buff<i32,3>,1> is a rank-1 stream with a rank-3 buffer of 32-bit integers.

Operations and Functions: The current STeP implementation comprises 14 operations and three functions, providing a simple yet highly expressive interface to manipulate streams. We started with an initial set of operations inspired by Parallel Patterns [9] and added operations for better expressiveness. Although Table 1 only introduces a subset of the STeP operations, STeP has other operations and functions, such as shape manipulation (e.g., *Reshape, Flatten, Promote, Window, Permute*), index generation (e.g., *Range, Enumerate*), buffer and stream conversion (e.g., *Bufferize, Streamify*), or stream adjoining (e.g., *Zip*). Similar to *Map, Flatmap*, and *Accum*, there exists other higher-order operations that receive a function as input (*e.g., Scan*).

2.2 A STeP Program Example

Figure 1 (a), shows how the dynamic operations can express Mixture of Experts (MoE) layers [13]. Partition receives a stream from the routing network and enqueues the input to selected experts. Reassemble collects the outputs by only dequeuing from the selected experts. To make the collection happen in the same order it has been sharded, it shares the same control signal with Partition. We also show a program graph for row-wise Softmax in Figure 1 (b). While the input stream encodes a $N \times N$ matrix, STeP does not require saving the $O(N^2)$ intermediate matrices to on-chip memory and only requires an $O(N)$-sized FIFO between the first Map and Zip; This is because STeP only stores the intermediate data while Accum reduces over each row.

2.3 Future Research Direction

We intend to investigate mapping STeP to streaming dataflow accelerators. By connecting operations and FIFOs with proper depths in the mapping process, we envision our abstraction can help execute various forms of dynamism on streaming dataflow accelerators while retaining high utilization.

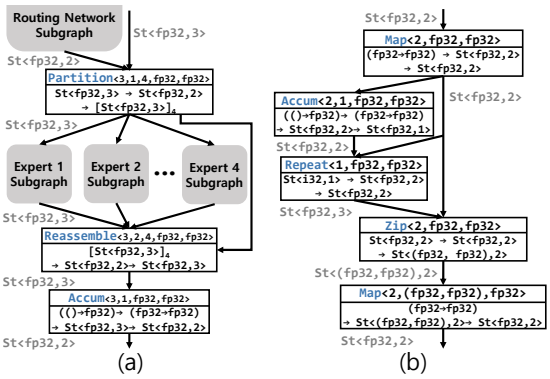

Figure 1. STeP program for MoE (a) and Softmax (b)

3 Evaluation Methodology

First, to show STeP's expressiveness to write machine learning models, we will implement the latest MLPerf Inference Benchmark Suite [11]. Next, we will implement primitives in Parallel Patterns and StreamIt using STeP's operations to show that STeP does not lose its expressiveness compared to existing dataflow programming abstractions. Lastly, we will express models, such as Swith-Transformer [4] and OpenMoE [5], that use the MoE layer to simulate dynamic computation graphs and tensor shapes in STeP. To evaluate the correctness of our target workload implementation, we will build a simulator for each operation using a cycle-accurate functional simulator framework.

References

[1] Alex Carsello, Kathleen Feng, Taeyoung Kong, Kalhan Koul, Qiaoyi Liu, Jackson Melchert, Gedeon Nyengele, Maxwell Strange, Keyi Zhang, Ankita Nayak, et al. Amber: A 367 gops, 538 gops/w 16nm soc with a coarse-grained reconfigurable array for flexible acceleration of dense linear algebra. In *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pages 70–71. IEEE, 2022.

[2] Zhengyu Chen, Dawei Huang, Mingran Wang, Bowen Yang, Jinuk Luke Shin, Changran Hu, Bo Li, Raghu Prabhakar, Gao Deng, Yongning Sheng, et al. Ai soc design challenges in the foundation model era. In *2023 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–8. IEEE, 2023.

[3] Vidushi Dadu, Jian Weng, Sihao Liu, and Tony Nowatzki. Towards general purpose acceleration by exploiting common data-dependence forms. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 924–939, 2019.

[4] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

[5] Xue Fuzhao, Zheng Zian, Fu Yao, Ni Jinjie, Zheng Zangwei, Zhou Wangchunshu, and You Yang. Openmoe: Open mixture-of-experts language models, 2023.

[6] Olivia Hsu, Maxwell Strange, Ritvik Sharma, Jaeyeon Won, Kunle Olukotun, Joel S. Emer, Mark A. Horowitz, and Fredrik Kjølstad. The sparse abstract machine. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS 2023, page 710–726, New York, NY, USA, 2023. Association for Computing Machinery.

[7] Quan M Nguyen and Daniel Sanchez. Fifer: Practical acceleration of irregular applications on reconfigurable architectures. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1064–1077, 2021.

[8] Raghu Prabhakar, Sumti Jairath, and Jinuk Luke Shin. Sambanova sn10 rdu: A 7nm dataflow architecture to accelerate software 2.0. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 350–352. IEEE, 2022.

[9] Raghu Prabhakar, David Koeplinger, Kevin J. Brown, HyoukJoong Lee, Christopher De Sa, Christos Kozyrakis, and Kunle Olukotun. Generating configurable hardware from parallel patterns. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, page 651–665, New York, NY, USA, 2016. Association for Computing Machinery.

[10] Raghu Prabhakar, Yaqi Zhang, David Koeplinger, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. Plasticine: A reconfigurable architecture for parallel paterns. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 389–402, 2017.

[11] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459. IEEE, 2020.

[12] Alexander Rucker, Shiv Sundram, Coleman Smith, Matthew Vilim, Raghu Prabhakar, Fredrik Kjolstad, and Kunle Olukotun. Revet: A language and compiler for dataflow threads. *arXiv preprint arXiv:2302.06124*, 2023.

[13] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[14] William Thies. *Language and Compiler Support for Stream Programs*. Ph.d. thesis, Massachusetts Institute of Technology, Cambridge, MA, Feb 2009.

[15] Yaqi Zhang, Nathan Zhang, Tian Zhao, Matt Vilim, Muhammad Shahbaz, and Kunle Olukotun. Sara: Scaling a reconfigurable dataflow accelerator. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 1041–1054, 2021.