Dynamic Memory for Distributed Architectures

Abstract

The ever-increasing scale of applications, such as large language models (LLMs), has driven the development of highly distributed architectures such as Reconfigurable Dataflow Accelerators (RDAs). While such architectures have powerful computing capabilities and high on-chip network bandwidths, their restricted dynamic capabilities make it difficult to efficiently handle the memory dynamism in LLMs, including dynamically allocated, extensible, and reclaimed memory. In this work, we propose a virtualization mechanism and race-free protocol for reclaiming buffers with shared references on RDAs. We also discuss plans for evaluating the efficiency of the proposed mechanisms and future directions for virtualization across multiple distributed memory units and memory consistency models for RDAs.

Alloc. & Reclaim	Static		Dyn. w/o shared ref.		Dyn. w/ shared ref.	
Size	Static	Dyn.	Static	Dyn.	Static	Dyn.
SARA [29]	✔					
Revet [23]	✔		✔			
Our work	✔	✔	✔	✔	✔	✔

Table 1. Memory dynamism capabilities of prior work.

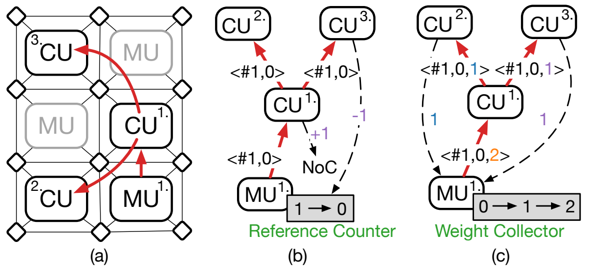

Figure 1. (a) Mapping a program to RDA, (b) Possible race condition in reference counting, and (c) Partial-free protocol.

1 Introduction

The advent of hyper-scale applications such as large language models (LLMs) has driven increasingly distributed architectures at all levels of the system, from runtimes such as Pathways [1] to accelerators such as Reconfigurable Dataflow Architectures (RDAs) [21]. Many algorithmic improvements for these applications, including speculative decoding [16, 18], KV-caching [14, 25], and Mixture-of-Experts (MoEs) [11, 24], are underpinned by dynamic control flow and memory. In this work, we propose several mechanisms that enable various types of memory dynamism on RDAs, namely dynamically allocated memory, dynamically extensible memory, and dynamically reclaimed memory. These concepts are roughly equivalent to the software ideas of malloc/ free [2, 15], STL vector [7, 13], and Garbage Collection (GC) [4, 12].

RDAs, such as Plasticine [21] and Sambanova's RDU [5, 20], are typified by a heterogeneous grid of compute units (CU), memory units (MU), and other tiles connected by a high-bandwidth Network-on-Chip (NoC) [28] (Figure 1 (a)). The distributed nature of these architectures enables compute capabilities on the scale of tens to hundreds of TFLOPs and on-chip network bandwidths in excess of 150 TB/s. In exchange, these architectures are generally equipped with restricted dynamic capabilities (Table 1). Prior work allocates statically-sized buffers to each MU and supports either static or dynamic allocation without shared references.

However, only allocating statically-sized buffers makes it difficult to efficiently handle optimizations where the total required size for the buffer is unknown at allocation time. For example, KV-caching has cached keys and values that continuously grow. In MoE, the size of the input to each expert is only known during runtime. Furthermore, the lack of shared references in dynamic memory allocation leads to inefficient memory usage, as the data must be copied for each reference. Supporting shared references is crucial for MoE as multiple experts execute on the same input data.

Dynamic memory reclamation for RDAs is an on-chip concurrent GC problem. However, common GC algorithms, such as tracing [4, 17] and reference counting [6], in RDAs can introduce race conditions or high overhead. For example, determining the reachable set in tracing requires sweeping all the pipeline registers of multiple CUs. Counter-based reference counting can suffer race conditions as shown in Figure 1 (b). CU1 sends a counter update (+1) to the MU since a new reference is created. Then, CU3 destroys the reference and sends a counter update (−1). However, the +1 update can occur after −1 due to delays in the NoC, and the MU will free the buffer while CU2 still holds a reference to it.

In this work, we introduce the mechanisms to enable two forms of memory dynamism on RDAs: First, we propose **buffer-level virtualization** to enable dynamically-sized (extensible) memory. We borrow the idea of virtual memory [8], but virtualize at the granularity of individual buffers instead. This allows multiple dynamically-sized buffers to be stored in the same MU and avoids fragmentation by assigning a dedicated virtual address space for each buffer. Second, we propose **partial-free**, a race-free protocol for reclaiming buffers with shared references. As partial-free does not require updating the counter whenever a reference is created, it reduces the synchronization overhead and congestion around the MUs.

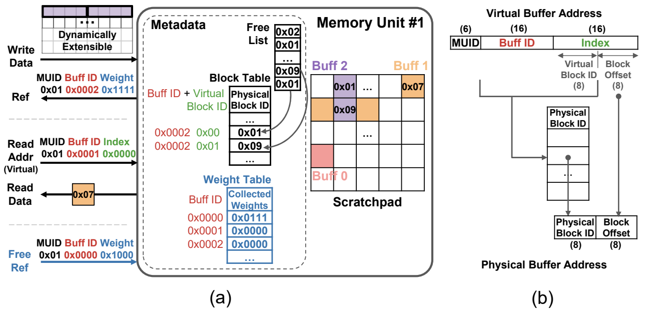

Figure 2. The design of the MU (a) and translation for virtualization (b). We assign value semantics to the buffers to decouple memory consistency from virtualization. The Weight Table is used to implement the partial-free protocol.

2 Virtualization

Our virtualization mechanism adds a layer of indirection by partitioning a MU's physical memory space into **Blocks** and record metadata using **Free List** and **Block Table**. The free list and block table manage free blocks and the virtual-physical block mapping, respectively. First, a dynamically extensible buffer gets allocated into the MU by creating a mapping between a buffer's virtual block and the physical block in the MU's scratchpad. As shown in Figure 2 (a), two buffers (Buff 0 and Buff 1) are already allocated in the MU. Then, the current buffer's ID will be `0x0002`, and MU will pull block `0x01` from the free list and create a mapping between the buffer's virtual block `0x00` and physical block `0x01`. After storing the buffer, the MU sends out a reference, and other units can use the memory unit ID (MUID), buffer ID (Buff ID), and the virtual index (Index) to read the buffer.

To show an example, we use parameters from Plasticine and add a few assumptions. As shown in Figure 2, MUID needs 6 bits since Plasticine has 64 MUs. The physical block ID and block offset take up 16 bits as a single MU's size is 256KB, and data is accessed in 4-byte words. The buffer ID needs 16 bits because we assume the smallest buffer size is 4 bytes, which means the MU can have at most 64K buffers. The virtual block ID and the block offset use 16 bits, assuming that a single buffer will not exceed a MU.

3 Partial-free

The partial-free protocol is inspired by weighted reference counting [3, 27]. Each reference is assigned a weight instead of tracking the number of live references. MUs atomically add the received weights and releases the buffer when the collected weight reaches the total weight.

In Figure 1 (c), MU1 emits a reference to Buffer 0 with the total weight (2), and the collected weight in MU1 is set to `0`. When the reference reaches CU1, the weight (2) is split as a new reference is created. CU2 and CU3 will return the weights to MU1 once they finish accessing the buffer. MU1 atomically adds the received weights for each buffer in the Weight Table shown in Figure 2. When MU1 detects that the collected weight reaches 2, it releases Buffer 0.

4 Evaluation Methodology

4.1 Workloads and Tools

Workloads To simulate behaviors such as dynamically allocated, extensible, and reclaimed memory, we will use models that use the MoE layer such as Mixtral 8x7B [11] and Switch-Transformers [9]. We will apply speculative decoding and KV-caching to models such as GPT [22] and Llama [26].
Tools We will implement the dynamic memory management policies on an RDA hardware model using a parallel cycle-accurate simulator. Then, we will augment the mapping algorithm of the SARA compiler to lower the targeted applications to the hardware model. We will also do design space exploration using a black-box optimizer [10].

4.2 Evaluation

Virtualization First, we will implement a baseline that reserves a certain amount of memory and reallocates and copies if the reserved space gets full. Then, we will compare the throughput and peak memory utilization improvements with the space and time overhead of implementing the virtualization. Second, we will measure the efficacy of allocating multiple buffers to a single MU by implementing the baseline that only allocates a single buffer to a single MU [23, 29] and measuring the improvement in peak memory utilization.
Partial-free First, we will compare the partial-free protocol with counter-based reference counting by measuring the throughput, latency, and synchronization overhead. To avoid the race condition shown in Figure 1 (b), we will gate the activation of CU2 and CU3 with a signal from the MU acknowledging that it has received the update sent from CU1. Second, we plan to explore the optimal design for the weight by varying the number of bits and splitting strategy. As this depends on the number of shared references in the program, we plan to conduct a static program analysis.

5 Future directions

One potential direction is expanding the virtual space of each buffer by adding another layer of indirection to map a buffer across multiple MUs. We can exploit the wide data path of RDAs (due to SIMD processing) and allocate more bits for the virtual index and metadata for the additional indirection. Another direction is the memory consistency model for RDAs. RDAs have unique constraints that require a different way of reasoning about memory orderings from CPUs [19]. Co-designing the proposed mechanisms and an efficient memory consistency model will be important to execute complex computation graphs efficiently.

References

[1] Paul Barham, Aakanksha Chowdhery, Jeff Dean, Sanjay Ghemawat, Steven Hand, Daniel Hurt, Michael Isard, Hyeontaek Lim, Ruoming Pang, Sudip Roy, et al. Pathways: Asynchronous distributed dataflow for ml. *Proceedings of Machine Learning and Systems*, 4:430–449, 2022.

[2] Emery D Berger, Kathryn S McKinley, Robert D Blumofe, and Paul R Wilson. Hoard: A scalable memory allocator for multithreaded applications. *ACM Sigplan Notices*, 35(11):117–128, 2000.

[3] David I Bevan. Distributed garbage collection using reference counting. In *PARLE Parallel Architectures and Languages Europe: Volume II: Parallel Languages Eindhoven, The Netherlands, June 15–19, 1987 Proceedings 1*, pages 176–187. Springer, 1987.

[4] Hans-J Boehm, Alan J Demers, and Scott Shenker. Mostly parallel garbage collection. *ACM SIGPLAN Notices*, 26(6):157–164, 1991.

[5] 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.

[6] George E Collins. A method for overlapping and erasure of lists. *Communications of the ACM*, 3(12):655–657, 1960.

[7] Damian Dechev, Peter Pirkelbauer, and Bjarne Stroustrup. Lock-free dynamically resizable arrays. In *Principles of Distributed Systems: 10th International Conference, OPODIS 2006, Bordeaux, France, December 12-15, 2006. Proceedings 10*, pages 142–156. Springer, 2006.

[8] Peter J Denning. Virtual memory. *ACM Computing Surveys (CSUR)*, 2(3):153–189, 1970.

[9] 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.

[10] Erik Hellsten, Artur Souza, Johannes Lenfers, Rubens Lacouture, Olivia Hsu, Adel Ejjeh, Fredrik Kjolstad, Michel Steuwer, Kunle Olukotun, and Luigi Nardi. Baco: A fast and portable bayesian compiler optimization framework. *arXiv preprint arXiv:2212.11142*, 2022.

[11] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[12] Richard Jones and Rafael Lins. *Garbage collection: algorithms for automatic dynamic memory management*. John Wiley & Sons, Inc., 1996.

[13] Nicolai M Josuttis. The c++ standard library: a tutorial and reference. 2012.

[14] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[15] Doug Lea and Wolfram Gloger. A memory allocator, 1996.

[16] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[17] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4):184–195, 1960.

[18] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*, 2023.

[19] Paul Mure, Nathan Zhang, Caroline Trippel, and Kunle Olukotun. Tags: A Framework for Distributed Event Ordering. In *Workshop on* the 2023 Programming Languages for Architecture (PLARCH), 2023.

[20] 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.

[21] 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.

[22] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[23] 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.

[24] 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.

[25] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR, 2023.

[26] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[27] Paul Watson and Ian Watson. An efficient garbage collection scheme for parallel computer architectures. In *International Conference on Parallel Architectures and Languages Europe*, pages 432–443. Springer, 1987.

[28] Yaqi Zhang, Alexander Rucker, Matthew Vilim, Raghu Prabhakar, William Hwang, and Kunle Olukotun. Scalable interconnects for reconfigurable spatial architectures. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, page 615–628, New York, NY, USA, 2019. Association for Computing Machinery.

[29] 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. IEEE, 2021.