

# The Case for Domain-Specific Networks

Dennis Abts  
NVIDIA  
dabts@nvidia.com

John Kim  
KAIST  
jjk12@kaist.edu

## ABSTRACT

Modern parallel computers are dichotomized into capacity or capability systems. Capacity systems cater to a wide range of weak scaling workloads, using distributed parallel systems with message passing while capability systems focus on strong scaling workloads across a significant fraction of the machine's processing units. The interconnection network differs under these regimes, with commodity Ethernet or Infiniband solutions typically deployed for capacity systems, while capability-class systems often necessitate tightly-coupled, fine-grained communication. Systems built for AI training and inference embody traits from both classes: tight coupling and strong scaling for model parallelism, and weak scaling for data parallelism in a distributed system. Handling 100-billion-parameter large-language models and trillion-token data sets presents computational challenges for current supercomputing infrastructure. This paper discusses the crucial role of the interconnection network in these large-scale systems, advocating for flexible, low-latency interconnects that can deliver high throughput at large scales with tens of thousands of endpoints. This work also emphasizes the importance of reliability and resilience in enduring long-running training workloads and demanding inference requirements of domain-specific workloads.

## I. INTRODUCTION

The advent of large language models (LLMs) such as GPT-4 has prompted the development of numerous variants aimed at reducing the costs associated with training and deploying these models on a large scale. The training of state-of-the-art LLMs demands substantial computational power distributed across thousands of GPUs over months of continuous training. When these model are used (for inference) by hundreds of millions of daily active users, the need for multiple systems, each with many nodes to handle incoming requests, becomes apparent. However, despite the focus on training these large-scale LLMs, deploying them poses even greater challenges. Meeting the expectations of end users, who anticipate a reasonable quality of service with response times of approximately 500 milliseconds to avoid perceptible “lag” and potential negative user experiences, is a complex task in itself. In order to make LLMs cost-effective, it is essential to emphasize the efficiency of both training and inference. This need has sparked the innovation of specialized systems tailored for particular domains, integrating either GPU or domain-specific processing elements with a domain-specific interconnected network [8][2][3].

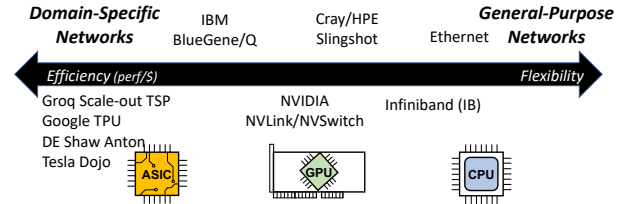


Fig. 1. High-level qualitative comparison of different networks that have been built.

Modern parallel computer systems consist of three main elements: processing units, memory & storage, and inter-processor communication or the interconnection networks. With the limit of Moore's Law scaling, the exploration of alternative architectures have ramped up to continue boosting performance and efficiency. Domain-specific architectures, or specialized hardware designed for specific applications, have recently emerged. At the same time, processing-in-memory architectures have been proposed, including domain-specific memory for LLMs [6], to improve both memory bandwidth and processing efficiency. Given these advancements, the interconnection network, which often is the bottleneck in system performance and scalability, must evolve to facilitate efficient, scalable systems. This paper champions the adoption of *domain-specific* networks as a key part of broader domain-specific system architectures. We outline the defining characteristics of a domain-specific network, review recent implementations of domain-specific networks, and conclude with potential pathways towards efficient networks for this novel system design approach.

## II. WHY DOMAIN-SPECIFIC NETWORKS?

### A. Different type of Networks

A summary of different type of networks that have been built is shown in Figure 1 and illustrates the diverse range of networks with “general-purpose” networks on one end of the spectrum (e.g., Ethernet) and domain-specific networks on the other end (e.g., Google TPU interconnect or Groq TSP scale-out network).<sup>1</sup> The key trade-off between the different type of networks weighs efficiency against flexibility, very similar to the trade-off in processor architecture with CPU vs GPU vs ASICs. The domain-specific networks often provide the highest efficiency or performance per unit cost as the network is a custom interconnect often with a custom protocol that

<sup>1</sup>The list is not exhaustive but provides a basis for comparison/discussion.

targets a particular domain or application. In addition, the network is often tightly coupled with the processing cores that it interconnects. However, such interconnect often cannot be leveraged for other domains with the same efficiency, even if the interconnection network organizations are very similar. For example, both the Google TPU [3] and the Anton2 [7] are based on a torus topology in their scale-out system. However, the interconnect for Anton is optimized for molecular dynamics application that has communication focused on nearest neighbor while the Google TPU interconnect is focused on global AllReduce communication for distributed training of machine learning models.

In comparison, a general purpose network, including Ethernet and Infiniband for high-performance computing, provides the most flexibility, supports unicast and multicast traffic with diverse communication characteristics, and most communication libraries build on top of these network’s abstractions and provide the ability to support different communication pattern including collective communications, such as AllReduce and All-to-All. Additionally, Infiniband networks, unlike Ethernet, have moved toward *adaptive* routing to avoid head-of-line (HoL) blocking that arises from congestion in oversubscribed fat-tree networks. Adaptive routing helps reduce the average packet latency, and more importantly, reduces the latency *variance* observed at the endpoints making collective operations (with an implicit barrier operation) more efficient since they do not incur as much delay waiting for the barrier (collective) to complete. Despite improvements, overhead of the communication protocol and encoding inefficiency at varying network layers acts like friction on the network performance and can limit the overall system performance.

Some of the networks used by the most powerful supercomputers are not necessarily general-purpose networks but do provide more flexibility compared to domain-specific networks. Examples include the IBM BlueGene/Q and Cray/HPE Slingshot networks that are based on proprietary low-latency interconnects for supercomputers but also provide an Ethernet-like interface to increase its compatibility and deployment flexibility. Similar to how GPUs provide more flexibility compared to dedicated accelerators and more efficiency compared to general-purpose CPUs, the interconnect for GPUs (e.g., NVIDIA NVSwitch/NVLink) represents a network that is somewhere in between domain-specific and general-purpose networks as it provides multiple virtual channels to enable support for various topology and flow control options. More importantly, it provides very high bandwidth necessary to enable efficient GPU-to-GPU communication of both training and inference workloads on multi-GPU systems [5].

## B. Characteristics of Domain-Specific Networks

The role of a domain-specific network is to support specific communication or data movement between the domain-specific accelerators at full data rate; thus, an oversubscribed general-purpose network may not necessarily be suitable. In the rest of this section, we summarize the key characteristics

of domain-specific networks and why they are necessary and present case study of recent domain-specific networks.

**Latency & bandwidth:** While reducing latency is important, bandwidth is often more important for domain-specific systems. However, general-purpose networks often cannot provide the necessary bandwidth required to interconnect the domain-specific accelerators together. In addition, the ability to tolerate high latency is often more important for domain-specific networks as the processing (compute) architectures are often throughput-bound and not necessarily latency-bound.

**Custom communication protocol:** While raw bandwidth is obviously important, it is also important to enable high utilization of the given bandwidth. The communication protocol of general-purpose networks cause high overhead, including packet headers, packetization overhead, as well as overheads from flow control and/or routing. To maximize utilization of the channel bandwidth, custom protocols are necessary to enable efficient domain-specific networks.

**Exploit (known) communication patterns:** Compared to general-purpose networks that need to support diverse traffic patterns and result in different utilization of the network, one key difference with domain-specific networks is that the communication pattern for a given domain or application is known in advance. As a result, the network can be designed and optimized to match the communication pattern, including the topology and routing within the system. In addition to the spatial traffic pattern, the temporal traffic pattern (e.g., burstiness) can also be exploited as well as the communication message size (e.g., short vs large messages) that is pre-defined, such as the tensor sizes.

**Low cost & complexity:** Cost of the network is always important; however, it is more important when all other components of the systems are carefully optimized for efficiency (e.g., performance per cost). The network cost not only includes the channels but also the network-interface cards, and the switches. The communication cost is also high if communication needs to occur through the host. As a result, many domain-specific networks often have *integrated* router or switch organization to avoid the need for introducing a dedicated switch and/or a dedicated network interface. This often leads to *direct* topology for domain-specific network where communication occur directly between the peer domain-specific architectures without communicating through the host. Another side-effect of known-communication patterns is that the traffic patterns are often “well-behaved” such that the amount of network or endpoint congestion is limited. Thus, complex congestion management is not necessarily needed and switch microarchitecture also need to be optimized to minimize complexity.

## C. Case Study of Domain-Specific Networks

High-bandwidth, low-latency interconnection networks are critical to scale-out systems but general-purpose networks have two fundamental limitations: 1) overhead from protocol and channel inefficiency, especially for small messages and 2) high overhead in launching communication kernels to the

accelerators. The performance overhead comes from the need to encapsulate messages into “packets” and out-of-order packet arrival/reordering complexity. Hardware congestion management and adaptive routing also lead to overhead and performance non-determinism. In this subsection, we describe some recent domain-specific networks that have been proposed, including the interconnect in Groq TSP and Google TPU supercomputer. We also describe the network for NVIDIA GPUs that is more generalized than domain-specific networks but achieves higher efficiency compared to general-purpose networks.

**1) Groq Scale-out Network:** The Groq Tensor Streaming Processor (TSP) [1] is an hardware accelerator for machine learning (ML) and high-performance computing. Unlike other accelerators, the compiler orchestrates program execution on a cycle-by-cycle basis so ML operations are performed by an ensemble of TSP instructions executing in parallel on different functional units within a single TSP as well as across multiple TSPs. Because the compiler comprehends the latency of each instruction and the transit time on stream registers, it informs hardware exactly what to do and when. This allows Groq TSP performance to be deterministic and predictable.

**Globally Deterministic Communication:** To scale a single-chip TSP’s deterministic programming model, a multi-TSP system must provide the *illusion* of communicating *synchronously*. The scale-out TSP allows efficient sharing of the global SRAM across different TSPs without requiring a mutex to guarantee atomic access to the global memory — instead, explicit *send* or *receive* instructions at *specific times* are used for communication and can reason about program correctness from this total ordering. The unique scale-out network provides deterministic communication across the multi-hop network to enable the compiler to have cycle-accurate knowledge of all data movement across the chip-to-chip (C2C) links.

**Software-scheduled networking:** With the support of globally synchronous interconnect hardware, the Groq system enables *software-scheduled networking* (SSN) paradigm where all contention for both functional units and network links are resolved at *compile-time*. The need for adaptive hardware has thus been removed – arbitration, back pressure, deadlock avoidance hardware are no longer necessary since *dynamic* contention for hardware resources will never happen.

**Deterministic Dragonfly:** The Dragonfly [4] topology is adopted to minimize the network diameter while enabling a collection of nodes to be used to scale the effective radix. However, unlike previously proposed Dragonfly, this work combined the Dragonfly topology with SSN to create a *deterministic Dragonfly* organization. To build the Deterministic Dragonfly, the C2C interconnect extends the single-chip TSP determinism to a multi-hop network through a “route tensors, not packets” philosophy by replacing dynamically routed packets in the network with scheduled tensors at compile time. This not only removes any packet overhead but also removes hardware components of non-determinism including switch arbiters and deep input buffers.

**2) Google TPU Interconnect:** The Google TPU system is an example of a domain-specific supercomputer [3] where both the software components (e.g., Tensorflow programming system, compiler, etc.) and the hardware including the chip architecture and the interconnect, were co-designed for machine-learning applications and offers improved scale, reliability, and performance. It has a number of features that make it uniquely capable, including:

**Collective Communication:** The interconnect was designed with emphasis on the AllReduce communication between the TPUs to combine the weights across all of the nodes as it represents the primary communication pattern between the TPUs. Dense packaging allows 64 TPUv4’s (along with a pair of host CPUs to control them) in a rack enabling systems that scale to 4096 liquid cooled TPUs and supports mixed-precision large-scale distributed training.

**Bandwidth optimized network:** To achieve bandwidth optimal performance for AllReduce (i.e., ring algorithm) while providing scalable topology organization, a 2D-torus topology was built for the TPU v2/v3 that enabled hierarchical AllReduce across the two dimensions. The high-bandwidth interconnect enabled synchronous (compared to asynchronous) training while minimizing tail latency to accelerate the communication phase of training. Since the communication phase is in the critical path, a fast interconnect that quickly reconciles weights across learners with well-controlled tail latencies is critical for fast training. To scale the system further for TPUv4 and provide additional bisection bandwidth, the 2D-torus was extended to a 3D-torus through a *hierarchical* system – using electrical channels for a group of  $4 \times 4 \times 4$  TPUs while exploiting optical circuit switching (OCS) to scale up to 4096 TPUs [3]. The OCS enables different variations of the 3D-torus to reduce network diameter while the system reliability can be achieved from the reconfigurability of the OCSes – allowing the system to tolerate some link failures and quickly route around them as the OCSes are simply optical fibers connected by tiny mirrors.

**Integrated router & dense system packaging:** The TPUv4 incorporates a router that supports a 3D torus and inter-core interconnect (ICI) – board-level network interconnecting the four (4) TPUs on each PCB as a  $2 \times 2$  mesh, with 16 external ICI links destined to other trays to construct the  $4 \times 4 \times 4$  3D mesh of 64 TPUs in each rack.

**3) NVIDIA NVLink/NVSwitch:** NVLink is an interconnect technology that allows multiple GPUs to work together seamlessly, enabling accelerated computing across a range of applications. It enhances bandwidth, reduces latency, facilitates shared memory access, and provides synchronization capabilities, all of which contribute to improved performance and scalability in multi-GPU systems. Some key features and benefits of NVLink include the following:

**Increased bandwidth, low latency, and scalability:** NVLink reduces communication latency between GPUs by providing a direct, dedicated connection. This low-latency communication is crucial for tasks that involve frequent data exchanges between GPUs, enabling efficient parallel process-

ing and minimizing delays. NVLink provides significantly higher bandwidth compared to PCIe. It allows for bidirectional communication between GPUs with a very high throughput, enabling faster data transfer and improved performance in multi-GPU configurations. NVLink also enables scaling of GPU resources, memory, and compute power, which is particularly beneficial in applications that require extensive parallel processing, such as deep learning, scientific simulations, and high-performance computing.

**Shared memory space (unified memory):** NVLink enables GPUs to have a shared memory space, allowing them to access each other's memory directly. This shared memory architecture facilitates efficient data sharing and reduces the need for data transfers over the interconnect, further improving performance in multi-GPU configurations. GPUs can utilize a unified memory space, which simplifies memory management and data sharing between GPUs. It eliminates the need for explicit data transfers between GPU memories, making it easier for developers to write parallel code and take advantage of multi-GPU systems.

**Synchronization and coherency:** NVLink provides mechanisms for synchronization and cache coherency between GPUs. This ensures that data remains consistent across multiple GPUs, even when they are working on different parts of a computational task. It enables efficient parallelization and collaboration between GPUs.

### III. CHALLENGES & OPPORTUNITIES

Future systems need to be scalable, equipped to handle a large number of processors in order to fulfill the requirements of larger models and their performance needs. The ideal design for scale-out systems should not only expand global memory capacity but also improve arithmetic performance. The construction of scale-out or large-scale systems, ranging from early supercomputers to domain-specific systems (e.g., Google's TPU, Tesla's Dojo [8]) typically employ a programming abstraction that masks the complexities of the underlying hardware. However, domain-specific networks also reveal some details of the underlying hardware, enabling opportunities to leverage its distinct capabilities to enhance efficiency and reliability. This becomes increasingly crucial as system sizes expand. We summarize this positional paper with some unique challenges and opportunities of domain-specific networks in this section.

**Challenges #1: One size does not fit all** The introduction of new hardware often requires a fresh compiler and system software to maximize the capabilities of these novel chips. Such proprietary chips are customized to manage specific workloads, and the specific tasks constitute a significant proportion of their overall workloads. The driving force behind the development of custom hardware is usually to enhance efficiency, cut costs, and exploit economies of scale. However, relying on proprietary technologies can be cost-inefficient; in particular, it remains to be seen if a domain-specific network can be efficiently leveraged for other domains while still providing high-performance efficiency.

**Challenges #2: Changes in domain algorithms** Domain-specific systems are crafted with a specific focus area in mind, yet the algorithms driving them evolve over time. For instance, models used for a variety of machine learning tasks are continuously advancing; the ones in use today might not be the ones that will be employed in the future. A key point to note is that these varying models can lead to different communication patterns and impact domain-specific network utilization. This poses a challenge as the networks might not be equipped to efficiently handle these new communication patterns.

**Opportunities #1: New boundary between software and hardware** The traditional computing stack, with the application at the top and the devices at the bottom, creates layers of abstraction that separate hardware from software components. However, in the era of domain-specific accelerators, optimizations across these layers are vital for maximizing both overall performance and system efficiency. Domain-specific networks present new opportunities for these cross-layer optimizations, consequently redefining the boundaries between the software and hardware for the interconnection networks. For example, compiler can optimize the traffic pattern by fusing layers (operators) together and sharding the model across neighboring nodes using model parallelism to exploit packaging locality and preserve as much communication locality as possible. The knowledge of the traffic pattern also provides system designers the chance to simplify and re-imagine the construction of interconnection networks, especially when their traffic and data dependencies can be determined at compile time. As a result, traditional hardware-oriented tasks such as routing and load-balancing, congestion management, and resource arbitration, can potentially be moved from the hardware to the software to improve network efficiency.

**Opportunities #2: Computation/communication boundary** Hiding or overlapping communication cost with computation is commonly used to minimize the impact of communication but domain-specific networks provide new opportunities to overlap communication – potentially through compute within the switch or shared compute logic for communication and computation. In addition, the resilience of the communication or the interconnection network (and its impact on computation) needs to be explored to understand its impact on system scalability.

### REFERENCES

- [1] D. Abts *et al.*, "Think fast: A tensor streaming processor (TSP) for accelerating deep learning workloads," in *ISCA*, 2020, pp. 145–158.
- [2] —, "A software-defined tensor streaming multiprocessor for large-scale machine learning," in *ISCA*, 2022, pp. 567–580.
- [3] N. Jouppi *et al.*, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," 2023.
- [4] J. Kim *et al.*, "Technology-driven, highly-scalable dragonfly topology," in *ISCA*, vol. 36:3, 2008, pp. 77–88.
- [5] B. Klenk *et al.*, "An in-network architecture for accelerating shared-memory multiprocessor collectives," in *ISCA*, 2020, pp. 996–1009.
- [6] Y. Kwon *et al.*, "Memory-centric computing with sk hynix's domain-specific memory," in *Hot Chips 35 Symposium*, 2023.
- [7] D. Shaw *et al.*, "Anton 2: raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer," in *SC'14*.
- [8] E. Talpes *et al.*, "Dojo: The microarchitecture of tesla's exa-scale computer," in *Hot Chips 34 Symposium*, 2022, pp. 1–28.