FLEET– Fast Lanes for Expedited Execution at 10 Terabits: Program Overview

Fred Douglis
Perspecta Labs

Seth Robertson
Perspecta Labs

Eric van den Berg
Perspecta Labs

Josephine Micallef
Perspecta Labs

Marc Pucci
Perspecta Labs

Alex Aiken
Stanford University and SLAC

Keren Bergman
Columbia University

Maarten Hattink
Columbia University

Mingoo Seok
Columbia University

***Abstract*—**
The DARPA FastNICs program targets orders of magnitude improvement in applications such as deep learning training by making radical improvements to network performance: while raw bandwidth has grown dramatically, the fundamental roadblock to application performance has been in delivering that data to the application. FLEET provides a primarily off-the-shelf solution with high-end servers and shared computational and storage resources connected via PCIe over a reconfigurable MEMS optical switch; it uses custom Optical NICs to allow arbitrary topologies that can be configured before or even during execution to take advantage of shared resources and to flow data between components. FLEET's software is derived from Stanford Legion, which we are modifying to use the FLEET hardware and to plan application execution for these dynamic network topologies.

Distribution "A" (Approved for Public Release, Distribution Unlimited). This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Introduction

In 2019, the U.S. Defense Advanced Research Projects Agency (DARPA) solicited proposals for a program on Fast Network Interface Cards (FASTNICS) [4]. Dr. Jonathan M. Smith created the four-year program in recognition of the large gap between processor speeds and optical networks on the one hand, and network interconnects on the other: network interfaces typically operate $100 - 1000\times$ slower than these other components, and this gap significantly impacts the performance of data-intensive applications. From the call for proposals:

> FASTNICS will speed up applications such as the distributed training of machine learning classifiers by 100x through the development, implementation, integration, and validation of novel, clean-slate network subsystems. The program will focus on overcoming the gross mismatches in computing and network subsystem performance.

This article reports the initial design of a platform for FASTNICS, called FLEET.[1] FLEET provides a primarily off-the-shelf architecture that can leverage continuing advances of commercial computing advances to meet or exceed the FASTNICS program goals. The project started in mid-2020, thus this is a design overview and status report rather than reporting a polished system. FLEET requires innovations in both hardware and software, which are described in greater detail in "Hardware Overview" and "Software Overview." We then discuss "Related Work" and report "Conclusions."

Hardware Overview

FLEET's key hardware innovations are *Optical Network Interface Cards* (O-NICs) that can be plugged into the Peripheral Component Interconnect Express (PCIe) slots to extend the PCIe communication channels into the optical domain at full PCIe bandwidth. PCIe in the optical domain allows fine-grained direct memory transfers between servers or devices without the disadvantages of a shared bus. Our choice of PCIe allows for extremely efficient, low overhead, transparent zero-copy Remote Direct Memory Access (RDMA) memory transfer between cooperating tasks, at aggregate speeds of 12 Tbps by the end of the program. Critically, our choice of PCIe networking also allows reconfigurable direct access to all standard PCIe device resources, such as Graphic Processing Units (GPUs), high performance Non-Volatile Memory Express (NVMe) storage drives, and data gathering sensors such as radar sensors, digital radios, and all other PCIe cards.

Once PCIe data is in the optical domain, we use a Micro-ElectroMechanical (MEMS) optical circuit switch to connect the FLEET O-NIC cards to each other, allowing full-PCIe bandwidth network communications between two servers, a server and a PCIe device, or between two PCIe devices. The O-NIC leverages work on photonic interconnects at Columbia University [3], [7].

Figure 1 provides an overview of the FLEET hardware architecture. Briefly, ① shows how multiple clusters can be interconnected via optical switches, with 3Tbps aggregate system-to-system throughput in the FLEET Generation 1 system (Gen1) and 12Tbps in FLEET Generation 2 (Gen2).

② shows the details for a single cluster. Each cluster nominally consists of four servers and up to six chassis to hold PCIe devices. All of these servers and devices are interconnected using optical fibers that carry the PCIe data channels using 16 wavelengths for the 16 PCIe lanes a single O-NIC supports. The fibers are connected to a Polatis Series 7000[2] 384x384 Software Defined Optical Circuit Switch, which configures mirrors and lightpaths (using Software Defined Networks) to allow the incoming and outgoing fibers to be connected to any other O-NIC port (flipping the incoming and outgoing signal paths). The 96 fibers not attached to servers or devices will be used for intercluster and WAN communication.

Each cluster is further broken out into ③ eight GPUs (or NVMe RAID controllers), each with its own O-NIC; and ④ eight CPUs, each with three O-NICs. ⑤ A single CPU consists of a sub-motherboard blade with 28 cores. Finally, ⑥ shows the O-NIC in detail. The use

[1] FLEET stands for **Fast Lanes for Expedited Execution at 10 Terabits**.

[2] https://www.polatis.com/series-7000-384x384-port-software-controlled-optical-circuit-switch-sdn-enabled.asp

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MIC.2021.3075326, IEEE Internet Computing

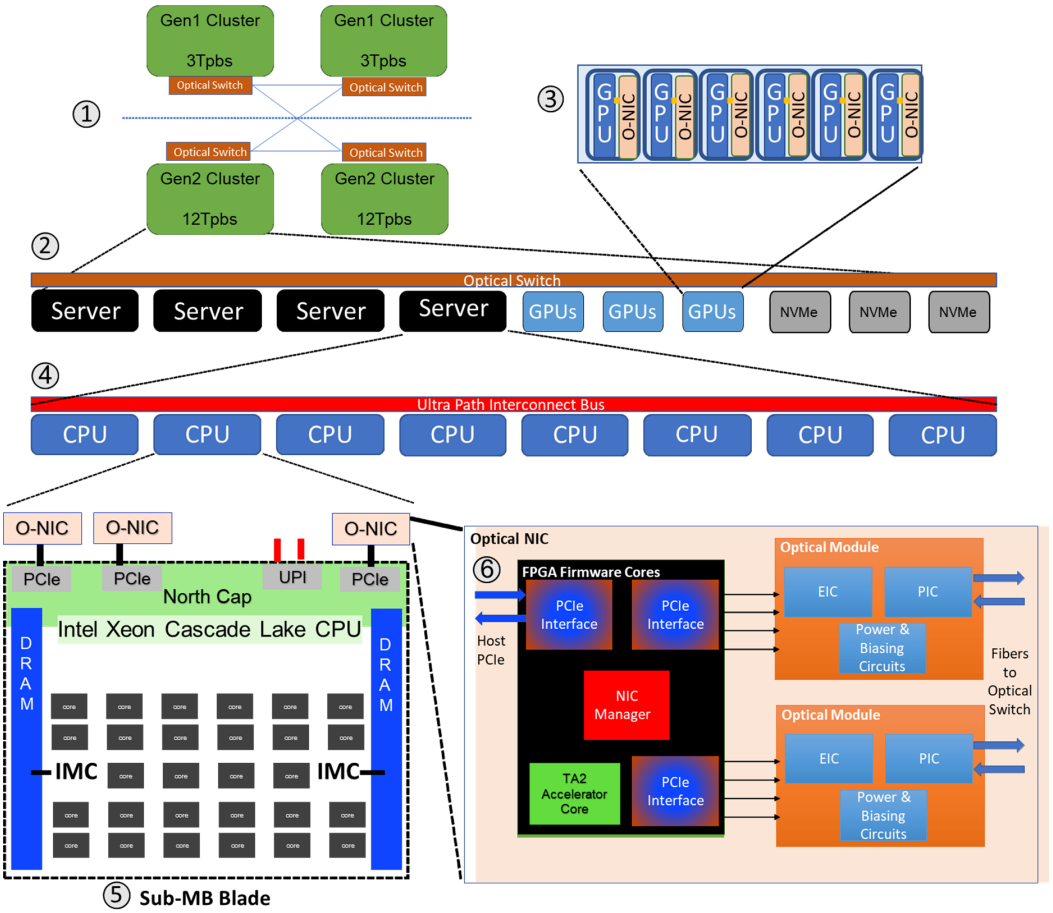

Figure 1. FLEET hardware overview

of PCIe to attach our network interface card is key to our hardware architecture because it will accelerate acceptance, deployment, and transition through use of proven, widely-available attachment standards. Further, PCIe is expected to have a long and productive roadmap for performance improvements. In addition, we are working toward improving compatibility, flexibility, and performance by extending FLEET to Lambda Labs A100 AMD systems[3] using PCIe generation 4.

The O-NIC has two key features:

1) The use of two optical-ports on the O-NIC card allows sophisticated data flows. With only one optical port, communications is restricted to only one peer, for example a GPU could use the O-NIC to read data directly from an NVMe storage device (meaning that the GPU could then not send its results to another stage in the processing pipeline without incurring a significant optical switch reconfiguration penalty). With multiple optical modules, the GPU can receive full bandwidth streaming input from the NVMe while sending full-bandwidth output to the next processing element. Multiple output pipes also allow efficient data distribution such as ring or tree network structures.
2) The O-NIC is more than a PCIe channel and two optical modules that convert the PCIe bits to photonics; it also has a sophisticated FPGA that runs a PCIe Manager firmware core providing security features, memory address translation, and virtualized PCIe devices to resolve the inherent problems of reconfigurable direct access to memory and physical PCIe devices. Beyond this, it also provides a general-

[3]https://lambdalabs.com/deep-learning/servers/hyperplane-a100

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MIC.2021.3075326, IEEE Internet Computing

purpose mechanism to allow in-network computation on data to accelerate latency-sensitive applications. Allowing arbitrary user applications to securely deploy code to the network FPGA interfaces minimizes the stage-to-stage latency where it is important; we use this functionality to meet the FASTNICS latency metrics. As detailed in "Related Work," the use of FPGAs for *in-situ* processing in network interface cards is by now well established, but to our knowledge the PCIe integration is unique.

In summary, FLEET provides a rapidly reconfigurable hardware architecture that can build custom hardware clusters from available LAN and WAN resources to execute specific applications, or application phases, with dynamically reconfigurable flexible pipelines providing direct access to shared resources. We provide a few examples of this innovative feature. All examples use the same underlying hardware platform, just with different applications and PCIe circuit switch configurations, effectively running a different hardware cluster:

- Simple two party exchange of data: Servers A & B can exchange data at 12Tbps.
- Direct data gathering. Server A can read radar sensor data from the radar PCIe data acquisition cards (or stored radar data from NVMe disks) at 12Tbps.
- Simple three party exchange of data: Server A can exchange data at 2Tbps with server B and at 10Tbps with server C. Any bandwidth ratio, in units of 126Gbps (the bandwidth of a single Phase 1 O-NIC), can be supported.
- Data multicast: Servers A, B, C, and D could output data onto a ring topology, allowing data generated at 12Tbps by server B to be received by C, D, and A. Equally well, server D could generate data to be received by servers A, B, and C at 12Tbps.
- Phased three-party exchange of data: Server A can exchange data at 12Tbps with server B. After the initial exchange needed for the first phase of processing is complete, server A can exchange data at 12Tbps with server C. While it is talking to server C, the switch may be reconfigured to allow seamless high-speed interaction with server D when it is done with server C.
- Processing pipeline using direct disk-GPU communications. Figure 2 depicts a number of CPUs, GPUs, and JBODs (just a bunch of NVMe disks), each with one or more O-NICs.[4] Here, the source image data from NVMe disks D_1 is sent directly (bypassing server CPUs and PCI Root Complex) to GPU G_1. After GPU G_1 has processed an image block, it forwards the output to server A which performs another stage of processing. After that stage of processing, the data can be sent to server C for processing, in conjunction with data from NVMe disks D_3, and with streaming FPGA transformations performed in O-NIC$_2A$. Similar pipelining occurs in the top half of the figure. All data transfers in this pipeline may be simultaneously performed at full PCIe bandwidth, allowing (if multiple disks and GPUs are used) 80 Tbps[5] of data to be in flight across the entire cluster at once provided the application could perform this streaming processing.

Software Overview

FLEET's software architecture builds on the Legion [1], [20] system, as shown in Figure 3. Legion is a well-established, open-source programming system for writing high-performance applications for distributed heterogeneous architectures. The blue components in the figure represent existing code, either in Legion or Linux. Red components represent FLEET code, including some new aspects of existing Legion code, as follows.

We originally anticipated that FLEET applications would be written in Regent [18], a language that supports implicit dataflow parallelism, or in C++. However, as the project evolved we shifted to Pygion [17], a flexible Python-based alternative to Regent, and in particular FlexFlow [10], a Legion application specifically focused on DNN Training. FlexFlow supports C++ and Python,

[4]The devices are labeled things like G_2 for GPU-2, or A for CPU A; the O-NICs are labeled to match their devices except for CPUs, which have multiple O-NICs and subscript which O-NIC is referenced, as in O-NIC$_1A$.

[5]The example of 80 Tbps is not a maximum limit: using more servers in the local cluster or across the WAN to other clusters could increase this simultaneous in-flight bandwidth usage arbitrarily.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MIC.2021.3075326, IEEE Internet Computing

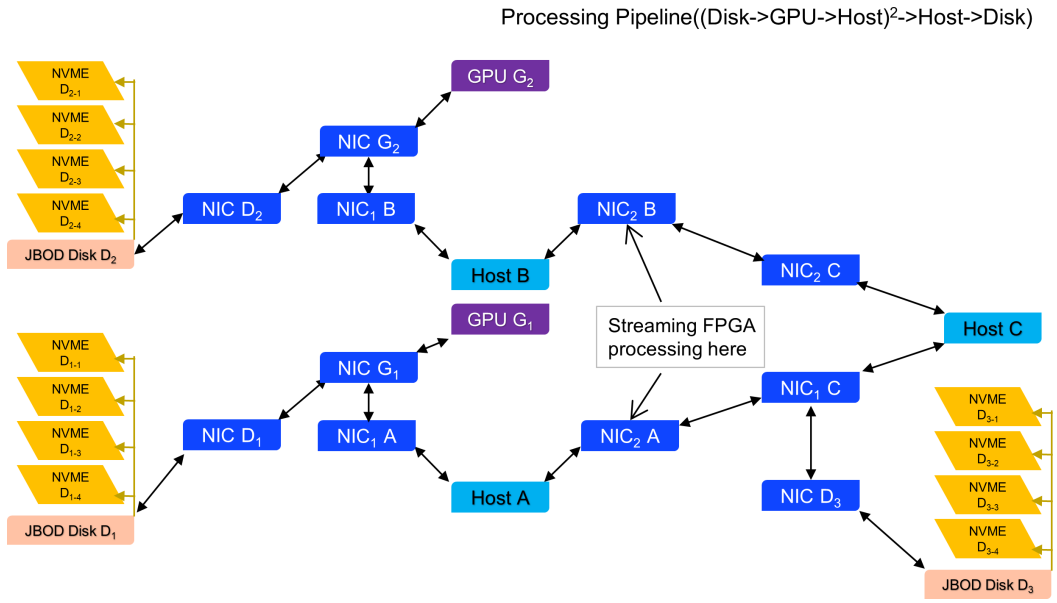

Figure 2. Sample topology

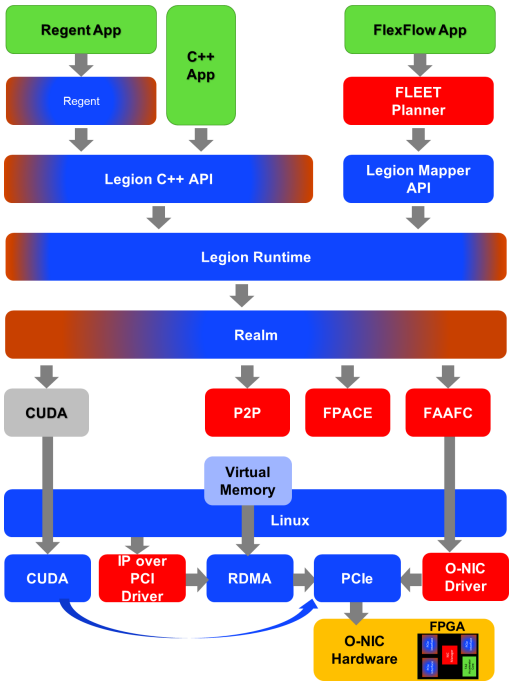

Figure 3. FLEET software overview

and it can support both Keras (TensorFlow) and pyTorch applications. It can work directly with the FLEET *Planner*, described in greater detail below, to allocate resources. The Planner uses a FlexFlow strategy interface and Legion *mapper* interface to control where application components (eg. DNN layers or operators) run; it is also responsible for determining the best network configuration to connect O-NICs through the MEMS switch, and for adapting that configuration at runtime.

Internally, Legion has a low-level portability layer called Realm [20] that sits below a high-level runtime (which we refer to as the Legion runtime). Both the high-level runtime and Realm need some modifications for the FLEET environment. For example, we are adding a transport library to Realm that supports our FPGA O-NIC communication protocol for message passing and RDMA. It would also be possible to add support for FLEET to a lower-level transport component, such as GASNet [2] or UCX [16], but after exploring those alternatives, we decided it would be simpler and more efficient to support FLEET directly in Realm.

Underneath the runtime, we have the FLEET *Planned Application Communication Environment* (FPACE), which manages data movement as tasks finish with stages of their processing. The *FPGA Application Acceleration Firmware Controller* functionality (FAAFC) permits applications to use the FPGA on the O-NIC as an external Legion processor, similar to how it offloads execution to GPUs.

The FLEET Linux drivers are shown at the

bottom of Figure 3, with most of them feeding into the PCIe driver and ultimately the O-NIC. The FLEET IP over PCIe Network Driver is responsible for transmitting IP packets over the PCIe, using the PCIe packet as the datalink layer in the IP stack. The O-NIC Driver is designed to recognize the O-NIC during Linux PCI Root Complex enumeration for discovery and support RDMA communication.

Regions are the primitive collection type in Legion; regions are a form of table similar to relations or dataframes, which are associated with a task and then migrate to another task for further processing. In the context of FLEET, these tasks may be on different cores within a sub-motherboard blade, on another blade, on another server, or even on another cluster. It is the responsibility of the runtime, using FPACE, to make the data available when it is needed, and to ensure coherence and performance.

FLEET Planner

The FLEET *Planner* creates the execution plan for the application using the FlexFlow strategy and Legion Mapper APIs. Legion already performs planning for NUMA considerations, memory and cache sizes, and heterogeneous computing elements (such as tasks that could be executed on GPUs or CPUs depending on what would provide the overall best system performance). However, our resource allocation problem is more challenging than the existing Legion mapping functionality of where to place data and computation. With FLEET, the PCIe channel bandwidth must be managed over time. FLEET is using an optical switch that can route PCIe slots to different destination, but changing destinations has an expensive reconfiguration time.[6] Thus, it is important for FLEET to plan the datapath through the optical system when it is placing code on processors to ensure that sufficient bandwidth is available for all purposes and that topology changes are seldom required and unobtrusive.

Accurate simulation of execution characteristics is a critical prerequisite to accurate planning. We have recently extended the FlexFlow [10] simulator to more accurately model Legion's runtime execution, including accounting for congestion on overloaded links. We have also made various improvements to the Legion profiler to more accurately report the costs of network messages and task execution on accelerators, which we expect will be important when we due detailed studies of application performance using O-NICs.

The Planner will create the network topology (the optical switch configuration assigning O-NIC communication circuits) and the assignment of tasks to servers, sockets, cores, GPUs, and FPGA Application Accelerator Firmware Cores. The FLEET Planner has the power to move the code to the data, the data to the code, or transform the data in-flight. If a particular application requires more resources (CPUs, GPUs, FPGA, or network bandwidth) than is available, the FLEET Planner will break the application up into phases where the network topology and resources can be reassigned to a new configuration to continue the application processing pipeline.

The FLEET Planner models the mapping problem as a graph partitioning problem, based on two graphs: $G_A = (V_A, E_A)$, the application task graph, and $G_T = (V_T, E_T f)$, the FLEET physical topology graph. We call the elements of V_A "vertices" (application compute tasks), and the elements of V_T "nodes" (compute nodes such as CPUs, GPUs, FPGAs, and the MEMS switch as well as memories (zero-copy, NVMe)). Nodes have attributes such as processor type and speed, memory type and capacity. The nodes in G_T are connected by edges $e \in E_T$ representing, e.g., UPI buses and PCIe channels. The presence of an edge expresses an affinity between *e.g.*, a processor and memory, or a processor and the MEMS switch. The edge has attributes such as bandwidth or latency (shared entities like Layer-2 cache and memory buses, are depicted as aggregated edges with shared attributes). The number of PCIe edges reflects the physical topology of the cluster: each O-NIC has two PCIe ports: $\Delta_{max,PCIe} = 2$. Similarly, the total number of PCIe edges in G_T connecting to the switch reflects the maximum number of ports on the switch, which is $\Delta_{max,switch} = 384$.

In order to run the application, the FLEET Planner needs to map each task vertex to a compute node, and route each data transfer along a shortest route in the physical topology graph

[6]Each reconfiguration of the MEMS optical switch takes many milliseconds, but the optical interfaces can take still longer before communication is reestablished.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MIC.2021.3075326, IEEE Internet Computing

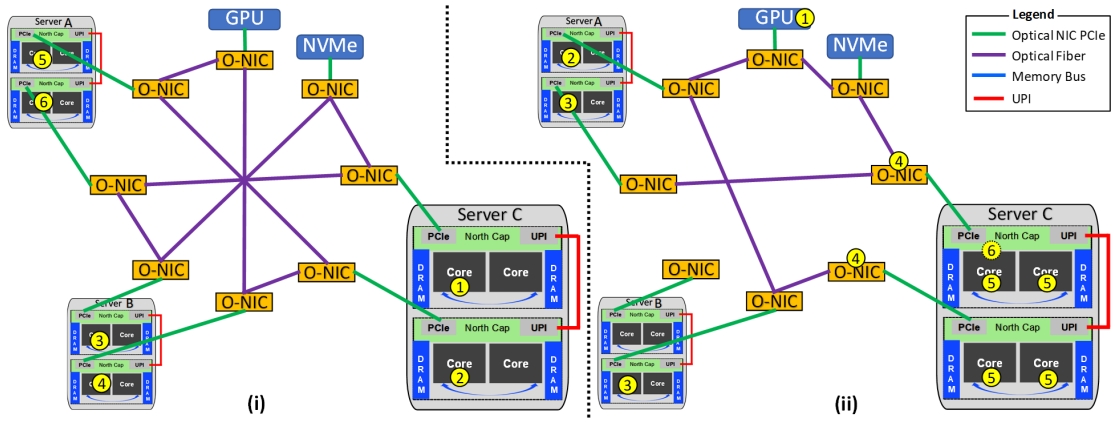

Figure 4. Notional FLEET Execution Plan: (i) Initial allocation of 6 tasks to cluster resources; (ii) Planner optimizes task allocation using cluster resources on cores, GPU, and O-NICs

from the compute node running its "source" task to the compute node mapped to its "destination" task. There is large preference to minimize the routing distance, e.g. 1-hop (direct connection) or even 0-hop (shared memory). For example, if these compute nodes are neighboring cores, then we can use page mapping so that the two nodes can use the same physical address. If the compute nodes are two CPUs in the same server, they can use the UPI on the motherboard that allows sending memory data between CPUs. If the source and destination are two servers, they can use the PCIe channels on the O-NIC to communicate. The FLEET Planner trades off bandwidth, latency, bus contention and cache contention, in order to optimize placement of tasks.

Figure 4 depicts the result of FLEET Planner processing. The initial chordal optical switch configuration and task mapping are shown in Figure 4(i). Figure 4(ii) shows the result of the Planner co-optimizing task placement and optical switch link configuration: it uses an alternate implementation of task 1 to execute on the GPU;[7] discovers that a direct link from the source data to the GPU running task ① is more efficient; as part of the processing pipeline, it sends the GPU results directly to task ② running on server A; the results of that task are forwarded simultaneously over the UPI bus to task ③ on server A and over the FLEET network to another copy of task ③ on server B; both task ③ results are forwarded to the Application Acceleration Firmware cores loaded on two of the O-NICs running task ④; the results of task ④ are forwarded to server C, where they are loaded in memory for both CPUs; task ⑤ executes on all cores of server C, taking advantage of the low latency costs for inter-core communication; finally, after all processing of task ⑤ is complete, task ⑥ finishes the application processing, also on server C.

FLEET's approach to jointly planning switch configuration and task mapping is to first find a good task partitioning independent of topology (switch) constraints, and find a good initial topology (meeting switch constraints) taking into account only structural information in the task graph. Second, we update the task mapping based on the "initial" topology. Third, we use iterative improvement (e.g. via stochastic search (e.g. Markov Chain Monte Carlo (MCMC) [6])) to improve on the initial solution, until the Planner solution is satisfactory or upon reaching a maximum iteration or time limit.

The initial task partitioning takes into account the application task graph G_A and hardware device graph. Since optimal scheduling is an NP-hard problem in general, so we will use heuristics and LP approximations in order to solve this approximately, based on the stated main objectives: balancing the application workload, and minimizing task communication cost.

FPGA Integration

To make use of the FLEET system, Legion needs two changes with respect to the FPGAs.

[7]This is a simplified example; in practice many GPUs would be used.

First, there needs to be a "shim" with enough of a Legion runtime executing on the O-NICs to be able to move memory regions without the involvement of the CPUs. Second, Legion needs to be augmented to treat FPGAs as an execution environment. Realm [20] provides a interface for adding new kinds of processors to the Legion runtime, which needs to support only a relatively small number of methods: launching a task on the device, triggering a finish event when a task has completed, transferring data to the device and transferring data back from the device are the main services that are required. (There are also methods associated with gathering profiling information, initiaizing the device on startup, and registering new tasks that can be launched on the device.) The main complication with FPGAs is that switching from launching a task A to a different task B can be very expensive if the FPGA needs to be re-flashed to load the code for task B. The cost of flashing is expensive, so ideally the FPGA can be programmed at the start of execution to include all functionality needed throughout a run; if the FPGA kernels do not fit or are more dynamic, the cost of flashing must be incorporated into both the Legion profiler and the simulator used for estimating FlexFlow costs.

Related Work

There is a long history of programmable network interfaces using FPGAs, e.g. P4 [8], and the programmability of the O-NIC is similar to recent work geared toward optimizing network throughput, such as Catapult v2 [5] and FlexNIC [11]. Catapult v2, from Microsoft, accelerated Bing searches via a "bump in the wire" inserting FPGAs between the NICs and the CPUs. FlexNIC allows the OS to add rules to packet processing in the NIC, which provide control over how packets are DMA'd to reduce memory pressure at high throughput. These research solutions also are generally restricted to fixed-purpose accelerator cores that do not change based on the application that is executing. Switching in datacenters has evolved in recent years; see [3] for a discussion.

While products from companies such as Dolphin [12], [13] permit servers to use PCIe to access memory and devices within other servers, that approach has significant restrictions: (a) Resource sharing requires double-PCIe bandwidth/lanes, latency, and server PCIe controller resources; (b) latency is high; (c) it cannot scale to 10Tbps (even theoretically) while still having "other" PCIe devices; and (d) there are no user-accessible computing resources on the PCIe Optical NIC. FLEET overcomes all these limitations.

High-performance custom-designed GPU fabrics such as NVLink/NVSwitch[8] interconnect GPUs (and sometimes CPUs[9]) with higher performance than PCIe. However, they are limited to short distances (typically an enclosure) and a fixed, and small, number of nodes. FLEET allows GPUs to be used by any CPU, interconnect directly with NVMe and other PCIe devices, and adapt arbitrary topologies through the O-NICs and MEMS switch.

There has been extensive work recently on training of deep neural networks of larger and larger size, including various pipelining techniques showing promising speedups by judiciously overlapping communication and computation [15], [14], [19], [9]. FLEET can add to these speedups by taking into account more fine-grained hardware details.

Conclusions

FLEET has been underway for less than a year as of this writing, so it is early in its R&D process. To summarize its capabilities:

- FLEET is a unique computing system that combines novel photonics interfaces (by Columbia University) with PCIe-based FPGA accelerators for orders of magnitude reduction in execution time of machine learning tasks. It is expected to reduce key workflows from days to minutes ($100\times$ improvement), not achievable on today's state-of-the-art commercial hardware (or their roadmaps).
- High bandwidth is important, but it's not enough. FLEET achieves low-latency by tightly integrating photonics and computing chips without the need for the high-level protocols used by state-of-the-art communication interfaces (e.g., Ethernet, RoCE, or Infiniband). It reduces overhead and latency and increases flexibility by using datacenter-scale PCIe as the

[8]https://www.nvidia.com/en-us/data-center/nvlink/

[9]One example is IBM POWER8 (https://www-355.ibm.com/systems/power/openpower/tgcmDocumentRepository.xhtml?aliasId=POWER8_with_NVIDIA_NVLink)

underlying data transport protocol, a standard ubiquitous protocol with continuous performance improvement in its roadmap.
- FLEET allows for a global pool of PCIe resources (GPUs, NVMe disks, or any other) to be used by different hosts and different workloads with zero overhead, unlike today's state of the art fixed-topology supercomputers or machine learning clusters, and
- it dynamically reconfigures its topology to provide efficient execution of machine learning workloads (and others).

We look forward to reporting additional progress as the project progresses.

REFERENCES

1. M. Bauer, S. Treichler, E. Slaughter, and A. Aiken. Legion: Expressing locality and independence with logical regions. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2012.
2. Dan Bonachea and Paul H Hargrove. Gasnet-ex: A high-performance, portable communication library for exascale. In *International Workshop on Languages and Compilers for Parallel Computing*, pages 138–158. Springer, 2018.
3. Qixiang Cheng, Sébastien Rumley, Meisam Bahadori, and Keren Bergman. Photonic switching in high performance datacenters. *Optics express*, 26(12):16022–16043, 2018.
4. DARPA. Broad agency announcement: Fast network interface cards (fastnics), August 2019. HR001119S0082.
5. Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking: Smartnics in the public cloud. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 51–66, 2018.
6. W.R. Gilks, S. Richardson, and D. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman & Hall/CRC Interdisciplinary Statistics. Taylor & Francis, 1995.
7. J. Gonzalez, A. Gazman, M. Hattink, M. G. Palma, M. Bahadori, R. Rubio-Noriega, L. Orosa, M. Glick, O. Mutlu, K. Bergman, and R. Azevedo. Optically connected memory for disaggregated data centers. In *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 43–50, 2020.
8. Ilija Hadžić and Jonathan M Smith. Balancing performance and flexibility with hardware support for network architectures. *ACM Transactions on Computer Systems (TOCS)*, 21(4):375–411, 2003.
9. Chaoyang He, Shen Li, Mahdi Soltanolkotabi, and Salman Avestimehr. Pipetransformer: Automated elastic pipelining for distributed training of transformers. In *https://arxiv.org/pdf/2102.03161.pdf*, 2021.
10. Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. In *Proceedings of the 2nd Conference on Systems and Machine Learning (SysML'19)*, 2019.
11. Antoine Kaufmann, SImon Peter, Naveen Kr Sharma, Thomas Anderson, and Arvind Krishnamurthy. High performance packet processing with flexnic. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 67–81, 2016.
12. Venkata Krishnan, Tim Miller, and Herman Paraison. Dolphin express: A transparent approach to enhancing pci express. In *2007 IEEE International Conference on Cluster Computing*, pages 464–467. IEEE, 2007.
13. Jonas Markussen, Lars Bjørlykke Kristiansen, and Hugo Kohmann. Nvme over pcie fabrics using device lending, 2019.
14. Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.
15. Jay H Park, Gyeongchan Yun, M Yi Chang, Nguyen T Nguyen, Seungmin Lee, Jaesik Choi, Sam H Noh, and Young-ri Choi. Hetpipe: Enabling large {DNN} training on (whimpy) heterogeneous {GPU} clusters through integration of pipelined model parallelism and data parallelism. In *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, pages 307–321, 2020.
16. Pavel Shamis, Manjunath Gorentla Venkata, M Graham Lopez, Matthew B Baker, Oscar Hernandez, Yossi Itigin, Mike Dubman, Gilad Shainer, Richard L Graham, Liran Liss, et al. Ucx: an open source framework for hpc network apis and beyond. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 40–43. IEEE, 2015.
17. Elliott Slaughter and Alex Aiken. Pygion: Flexible, scalable task-based parallelism with python. In *2019*

IEEE/ACM Parallel Applications Workshop, Alternatives To MPI (PAW-ATM), pages 58–72. IEEE, 2019.

18. Elliott Slaughter, Wonchan Lee, Sean Treichler, Michael Bauer, and Alex Aiken. Regent: A high-productivity programming language for hpc with logical regions. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, New York, NY, USA, 2015. Association for Computing Machinery.
19. Jakub Tarnawski, Amar Phanishayee, Nikhil Devanur, Divya Mahajan, and Fanny Nina Paravecino. Efficient algorithms for device placement of dnn graph operators. In *NeurIPS*, 2020.
20. Sean Treichler, Michael Bauer, and Alex Aiken. Realm: An event-based low-level runtime for distributed memory architectures. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, PACT '14, pages 263–276, New York, NY, USA, 2014. Association for Computing Machinery.

Acknowledgments

Thanks to our DARPA program manager, Jonathan M. Smith, for initiating the program and providing helpful guidance. We greatly appreciate the support of Zhihao Jia, Elliot Slaughter, and Sean Treichler, members of the Legion team not directly involved in FLEET. The project is a large team effort, including the FLEET team at Perspecta Labs, the Legion team at SLAC, the Lightwave Research Lab at Columbia, and our collaborators on the FASTNICS project at Raytheon and USC.

Authors

Fred Douglis is a Chief Research Scientist with Perspecta Labs. He has a background in distributed operating systems and resource management, storage, and other systems areas. He is a fellow of the IEEE and a member of the IEEE Computer Society Board of Governors. He served as editor in chief of *Internet Computing* from 2007-2010 and has been on its editorial board since 1999. Contact him at fdouglis@perspectalabs.com .

Seth Robertson is a Chief Research Scientist with Perspecta Labs. He has decades of expertise in large distributed systems, dynamic networks, cloud computing, DDoS Defense, avionics cyber defense, and deceptive, defensive, and offensive Computer Network Operations. Contact him at srobertson@perspectalabs.com .

Eric van den Berg is a Research Manager with Perspecta Labs. He has a background in applied mathematics, and has recently worked on distributed resource allocation in networks, planning and orchestrating cyber defensive maneuvers, and analyzing resource needs of quantum algorithms, among others. Contact him at evandenberg@perspectalabs.com .

Josephine Micallef is a Fellow and Senior Research Director of the Systems and Cyber Security Research group at Perspecta Labs. She is responsible for research initiatives on computing and networking platforms, technology, methodologies, and tools to support the construction and validation of large, complex, software-intensive distributed systems to ensure highly dependable operation even under cyber-attack. Contact her at jmicallef@perspectalabs.com .

Marc Pucci is a Chief Research Scientist with Perspecta Labs. He has worked on programs ranging in size from processor microcode to large-scale systems-of systems. He has developed a variety of operating systems including uni- and multi-processor, distributed, real-time, and embedded. Recent work includes the analysis of the power grid for inconsistencies, and the detection of aberrant device behavior using side-channel emissions. Contact him at mpucci@perspectalabs.com .

Alex Aiken is a Professor of Computer Science at Stanford University and the director of the Computer Science Division at SLAC. He leads the Legion project, which involves researchers from Stanford, SLAC, Los Alamos National Lab, and NVIDIA. He is a Fellow of the ACM. Contact him at aiken@stanford.edu .

Keren Bergman is the Charles Batchelor Professor of Electrical Engineering at Columbia University where she leads the Lightwave Research Laboratory and also serves as the Scientific Director of the Columbia Nano Initiative. Prof Bergman is a leading expert in high performance photonic interconnect systems with extensive experience leading project teams for DARPA, DOE, and ARPA-E. She is a Fellow of IEEE and OSA. Contact her at bergman@ee.columbia.edu .

Maarten Hattink is a graduate student with Columbia University. He received his B.S. and M.S. from the Eindhoven University of Technology, Netherlands, in 2015 and 2017. While pursuing these degrees he

worked at Prodrive Technologies B.V. as a software and FPGA engineer. He is now pursuing a Ph.D. degree and his research interest lies in photonic device integration and optical switching. Contact him at mh3654@columbia.edu .

Mingoo Seok is an associate professor in the Department of Electrical Engineering at Columbia University. He has been working on high-performance and low-power VLSI design with extensive experience in designing and prototyping analog, mixed-signal, digital, and power-management integrated circuits. Contact him at ms4415@columbia.edu .