

The Road Less Traveled: Eliminating Bottlenecks in High-Performance Computing Networking

Sinan G. Aksoy, Pacific Northwest National Laboratory (PNNL)


Roberto Gioiosa, PNNL

Mark Raugas, Laboratory for Physical Sciences

Stephen J. Young, PNNL

Scientific, engineering, and social real-life applications are often too large and complex to fit in a single workstation, both in terms of memory and computing requirements. Generally, a cluster of individual compute nodes interconnected by a high-performance network is required to solve such problems at the required scale. Ideally, such a system would function as if it were one huge computer, but in practice, because of the differences in access speed for local and remote resources, a complete new programming paradigm is required. In particular, because the access time difference between local and remote accesses could be in the order of 10–100x, it is paramount to effectively minimize and/or hide the latency of remote communication. Additionally, oftentimes multiple compute nodes need to access data on the same remote node (i.e., many-to-one communication patterns), causing network congestion and slowing down the entire application. As a consequence, one of the significant challenges in the use of modern cluster-based supercomputers is how to efficiently, robustly, and quickly handle the necessary communication between the nodes in the cluster. Both current and next-generation supercomputer designs have highly structured network topologies, such as the low-dimensional torus [1], fat tree [2], or DragonFly [3] topology, to have a straightforward routing scheme while attempting to mitigate the traffic congestion in high-communication applications. In many ways, these topologies have evolved and changed in lockstep with the message passing interface (MPI), the dominant programming model for distributed memory supercomputers, and have become tailored for particular classes of problems (i.e., numerical linear algebra and partial differential equations). However, even with modern high-performance network topologies, communication delays are often a significant bottleneck and dominate the overall computation time.

[Photo credit: iStock.com/carterdayne]

An aerial photograph of a city, likely Seattle, is shown with a dark, textured overlay. The image is covered with a pattern of binary code (0s and 1s) in various colors and orientations, creating a digital, data-driven aesthetic. The city's buildings, streets, and green spaces are visible through the semi-transparent overlay.

As a result of the interaction between the structure of internode communication in various classes of algorithms and the underlying network topologies, certain supercomputers gain a reputation for being more or less suited to a certain class of problems. Specifically, most state-of-the-art supercomputers have been optimized for traditional Linpack-style MPI applications which exchange large messages in highly structured (and often localized) patterns. However, as new problems have emerged that require high-performance computing (HPC) resources, for example, large-scale graph analytics and the training of machine learning models, being able to maintain performance on a more varied collection of communication paradigms has gained in importance. This is especially important to consider when executing on large HPC clusters is the only feasible option for modern graph analytics and machine learning workloads that show computation and memory requirements far beyond those available in a single workstation or small cluster. Of particular relevance to graph analytics and machine learning workloads is the communication performance of HPC systems when sending a large number of small, unstructured, and unpredictable messages. Furthermore, for many of these workloads, the communication patterns are only known at runtime as the computation evolves, making it impossible to predict and mitigate network congestion through smart data layout.

Rush hour and computing

The challenges faced by the HPC community can be understood, by way of analogy, through the evolution of urban transportation traffic. Consider, for example, the Seattle, Washington area. Seattle's arterial road networks, such as the I-5 and I-90 freeways, were developed during the "Boeing Boom." At this time,

the area's largest employers were geographically aligned with the natural traffic pipeline formed by the Puget Sound and Lake Washington. However, as new economic drivers emerged within Seattle, the city has become far more polycentric, with numerous hotspot destinations distributed throughout the region. Seattle's road network now has to contend with a daily influx of traffic from the surrounding Redmond and Bellevue into disparate parts of the city. The resulting traffic patterns are less predictable, less structured, and have (unsurprisingly) led to the development of at least 2,675 documented traffic congestion "hotspots."

While the design of communication architectures for HPC systems doesn't have the geographic limitations of traffic like the greater Seattle area, it is still influenced, much like the traffic network in Seattle, by decades of optimizations for a small class of traffic scenarios. Now that new unstructured traffic scenarios have become more prevalent, the old design paradigms are struggling to provide performance for these new workloads.

Fortunately, rather than having to repeat the decades of effort that went into optimizing HPC systems for MPI-style communications, the HPC communications can take inspiration from an industry that already had to deal with problems of unstructured communication—the telecommunications industry. As early as the 1970's, researchers at Bell Labs and IBM Watson Research Center were thinking about the problem of designing non-blocking switching networks in order to cost-efficiently scale telephone exchanges [4, 5]. Fundamentally, this is a question of how to effectively handle the unpredictable and unstructured telephone communication patterns. Eventually, this line of research coalesced around a single idea as being essential to handling

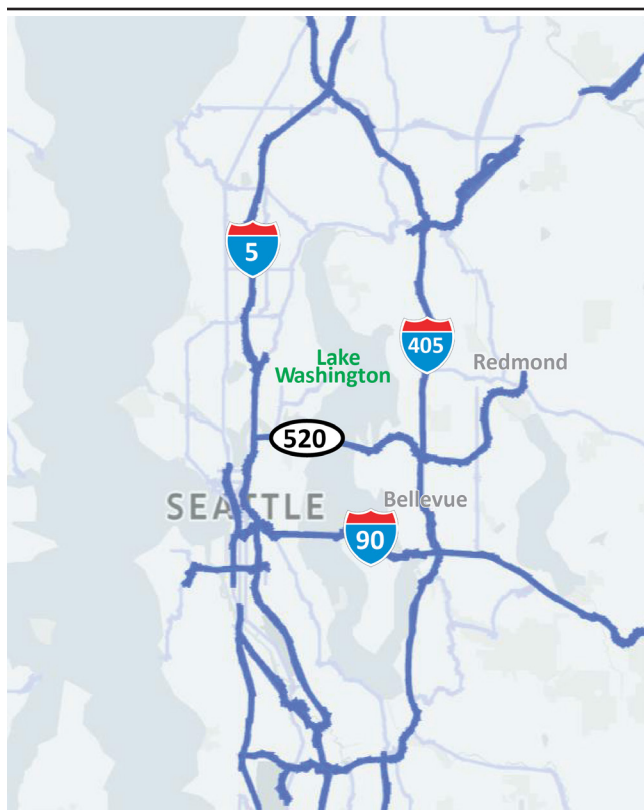


FIGURE 1. This map of the greater Seattle, Washington area road network with the major motorways (no stoplights) highlighted in blue shows lanes in each direction creating a bottleneck. Lake Washington, to the east of downtown Seattle, significantly impacts the topology of the road network, reducing the capacity and number of east-west routes throughout the region. In fact, between the two floating bridges (I-90 and WA-520), there are only five regular traffic lanes and two high-occupancy vehicle (HOV) lanes in each direction.

the unstructured communications of the telephone system—*expansion*. While many definitions of expansion have been proposed over the years, they all essentially reduce to the idea that the capacity of the connections leaving any local neighborhood scale with the size of the neighborhood. Returning to our analogy with Seattle traffic, we can see Lake Washington forms a fundamental obstruction to the expansion of the Seattle road network (see [figure 1](#)). No matter how you increase the capacity of the two floating bridges crossing Lake Washington, or even if you add new bridges crossing the lake, the capacity of the connection from Seattle to the east side will never be able to scale with the size of Seattle. Essentially, Lake Washington forms a geographic *bottleneck* and obstruction to expansion for traffic in

the Seattle area. Surprisingly, this fairly simple idea of considering networks with no bottlenecks has numerous practical applications from constructing circuits to efficiently perform matrix multiplication, to constructing codes which can effectively correct for errors, to methods to amplify weak sources of randomness to high-quality randomness suitable for practical randomized algorithms.

Given the wide applicability of networks with expansion [6], it is unsurprising that several communication topologies have been proposed which use expansion as a fundamental organizing principle. For example, both the Jellyfish [7] and Xpander [8] data-center architectures rely on expansion properties to provide a robust and extensible communication fabric. However, these topologies are fundamentally random in their construction which presents significant challenges in designing and validating the low-overhead communication schemes necessary in computational applications. In addition, the randomness of the connections presents significant obstacles to the adoptions of these topologies in HPC contexts.^a In fact, it is likely that the need for lightweight routing schemes (which are facilitated by highly structured topology) has led to the limited expansion properties of in-use and proposed HPC topologies [9]. However, there are known constructions which result in highly structured, optimal expanders [10]. The SpectralFly [11] topology, which we describe in the following section, is based on one such construction.

The infinite tree in the forest

Before describing the precise construction of the SpectralFly topology, it is helpful to think about exactly what a network with the best possible expansion (or alternatively, no bottlenecks) would look like. Returning to the traffic analogy, imagine traveling on a road network where every intersection is a four-way intersection. As you approach each intersection, you have four choices—turn around and go back along the road you were traveling on, or continue traveling on one of the other three road segments. Now if the road network has the best possible expansion, those three road segments must lead outside your “local neighborhood.” If we imagine continuing along this road network, at each intersection this repeats—you can either turn around or take one of three road segments which leave your “local neighborhood.” But as a consequence, the only way

a. In fact, one of the original proposers of the Jellyfish topology, Brighten Godfrey, obliquely referred to this challenge on his blog *You Infinite Snake*, writing “At this point, one natural reaction is that a completely random network must be the product of a half-deranged intellect, somewhere between ‘perpetual motion machine’ and ‘deep-fried butter on a stick.’”

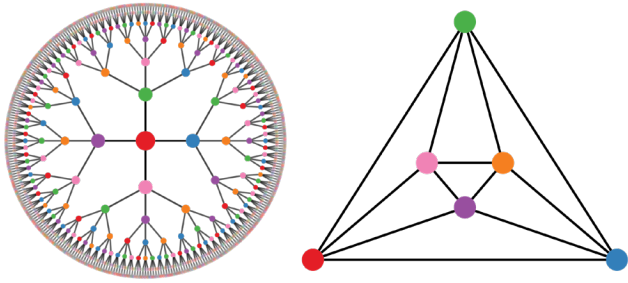


FIGURE 2. (Left) In this optimal expander, every vertex has exactly four connections. Since optimal expanders have no cycles, this unique optimal expander ends up being the four-regular infinite tree. **(Right)** The vertex colors in this vertex-edge graph for the octahedron are used to generate the vertex colors in the infinite tree (on the left) based on the traversals of the octahedral graph.

to return to an intersection you have already visited is to turn around and go back the way you came. In essence, if the road network has the best possible expansion properties, it must be the four-regular infinite tree (see [figure 2](#)).

Obviously, building an infinite tree to use as a road network or as an HPC topology is physically and financially impossible, but taking a slightly different viewpoint on the infinite tree can still provide considerable insight into the properties of networks with good expansion. Specifically, instead of considering the connections in the infinite tree to be physical, we can think about them as a record of decisions made. For example, if we were at the Space Needle in Seattle and wanted to go pick up a coffee at the original Starbucks located at the Pike Place Market, we could either go southwest on Broad Street, turn left on Western Avenue, and continue until we arrived at the Starbucks, or we could go east on Denny Way, turn right on Westlake Ave, and take a right on Stewart Street. While both of these routes will get us some much needed coffee, they emerge from a different sequence of decisions and so would be depicted as different vertices on the infinite tree.

In order to keep track of which locations are the same, we can color individual vertices to encode their location. We see this illustrated in [figure 2](#) where the coloring of the vertices in the infinite tree correspond to the “road network” depicted to the right that has six intersections and 12 roads. For example, in the finite network, the red vertex is adjacent to the green, pink, purple, and blue vertices, and we see that in the infinite tree, every vertex that is colored red is adjacent to a green, pink, purple, and blue vertex. In fact, the correspondence goes deeper than that, as the

colored infinite tree is simply a recording of all the potential routes through the finite network. Indeed, if we start at the red vertex in the finite graph and go to the green vertex, then the pink vertex, and back to the red vertex, in the infinite tree we end up at one of the red vertices in the upper portion of the image of the infinite tree. If, on the other hand, we go to the pink vertex, then the purple, and back to the red vertex, in the infinite tree we end up at one of the red vertices toward the bottom of the infinite tree, despite ending at the same vertex in the finite network. Thus, in many ways, the question of how to design networks with good expansion properties reduces to a perhaps simpler question: *how do you color the vertices of the infinite tree to preserve the expansion properties of the tree?*

To understand what such a coloring looks like, let us consider walking randomly around Seattle. In order to keep track of where we are, imagine every intersection to the west of Lake Washington is colored a different shade of blue, and every intersection to the east of Lake Washington is colored a different shade of red. Since Lake Washington is such a strong bottleneck, it is easy to see that if we start at a blue intersection we should expect to stay on blue intersections for a long period of time. But now think about what this means for the associated colored infinite tree—if we start at a blue vertex, as we go away from that vertex we should typically stay at blue vertices. But there are only so many blue vertices we can use, so that means that the infinite tree must be repeating shades of blue as it grows. In fact, this provides pretty good intuition for comparing two colorings of the infinite tree—a coloring is better at preserving expansion when it is more colorful than another coloring.

Given this framing, it is perhaps not surprising that randomly coloring the vertices of the infinite tree is an effective means of generating graphs with good expansion properties. In fact, this is the approach that is used by the Xpander and Jellyfish topologies to design high-performance data centers. However, this approach has significant drawbacks for HPC needs in that the lack of readily apparent structure in the resulting network means that significant effort needs to be spent in deciding the route any particular communication takes. Providing an explicit means of coloring the vertices of the infinite tree which—in some sense—preserves as much of the expansion property as possible, proved to be a significantly harder challenge.

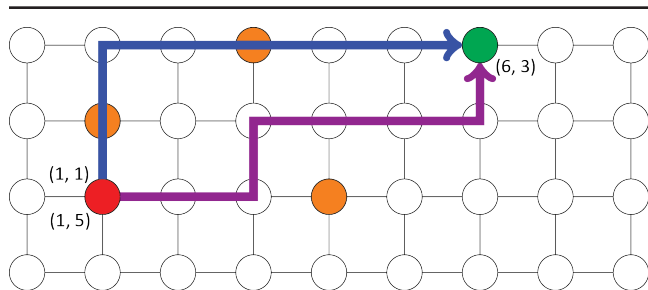


FIGURE 3. This figure depicts an idealization of a downtown street network in a grid-like area such as Manhattan, New York. The blue arrow represents the natural path one would take when going from the red intersection to the green intersection, whereas the purple arrow would be one possible path to avoid construction at the orange vertices.

To understand this challenge, it is helpful to return to the problem of navigating around a road network. However, this time instead of focusing on the freeway system, we will focus on navigating around downtown—perhaps some place like Manhattan, New York, where there is a strong grid-like structure such as shown in [figure 3](#). Imagine your friend calls you from the red intersection looking for directions to your favorite coffee shop, conveniently located on all four corners of the green intersection! How would you tell them to get there? You would probably say something like, head north for two blocks and then go east for five blocks. Or perhaps if you knew they were repairing the sidewalks at the orange intersections, you would tell your friend to go east for two blocks, head north for one block, go east for another three blocks, and finally head north for one more block. Now if the picture in [figure 3](#) was instead a diagram of the switches in an HPC topology and you were providing instruction on how to send information from the red switch to the green switch, you would likely express this idea differently (computers not being particularly well known for knowing which way is north, south, east, or west!). Perhaps you would give each switch a name, say the red switch is switch (1, 1) and the green switch is switch (6, 3), and then you would tell the switches to send the information out the port that increases the second coordinate twice, and the first coordinate five times. In fact, most modern and historical HPC topologies can be thought of in this light. Each switch has a “name”—often a vector of integers—and information is routed by performing a sequence of operations on these names, for example increasing or decreasing a coordinate. Oftentimes, there are additional rules which say two different names are effectively the same. For instance, if we

were to imagine connections between the top and bottom row of vertices in [figure 4](#), we would want to say that (1, 5) is an alternative name for the red vertex, as starting there and increasing the second coordinate four times would return us to the red vertex. Thus, the real challenge is to design a naming scheme for the infinite trees and operations on those names which maximize the colorfulness of the infinite tree.

In the late 1980’s, Lubotzky, Phillips, and Sarnak [[12](#)], and independently Margulis [[13](#)], provided a relatively simple naming scheme and set of operations to provide an optimal coloring scheme for a wide range of infinite trees and number of colors. The collection of names for every vertex is a list of two-by-two matrices with integer entries, and the operation going from one name to another is multiplication by one of a handful of two-by-two matrices. The SpectralFly topology is defined by using one of these networks as the interconnection network between the switches and then placing an appropriate number of compute nodes at each switch (see [figure 4](#)).

The fact that the colorings proposed by Lubotzky, Phillips, and Sarnak [[12](#)] are the best possible at preserving the expansion properties of the infinite tree relies on a deep result in the representation theory of automorphic forms originally conjectured by Ramanujan [[14](#)]. However, we can gain some intuition as to why their rules result in more colorful trees by comparing the operations with other topologies. For example, since the operation for the torus topology is incrementing/decrementing individual coordinates, the end location depends only on the number of increments/decrements per coordinate, not the particular order they are applied. In contrast to this, the results of matrix multiplication (in general) rely on the order of operations. That is, by applying the same set of operations in two different orders, it is possible to arrive in different locations. In [figure 5](#) we can see the difference in colorfulness in the infinite tree for torus topology and the SpectralFly topology.

Structural comparison with SpectralFly topology

The SpectralFly topologies promise as supercomputing topology is evidenced by its exceptional structural properties. We now put these properties in perspective, by comparing them against those of two well-known topologies: a DragonFly network and a

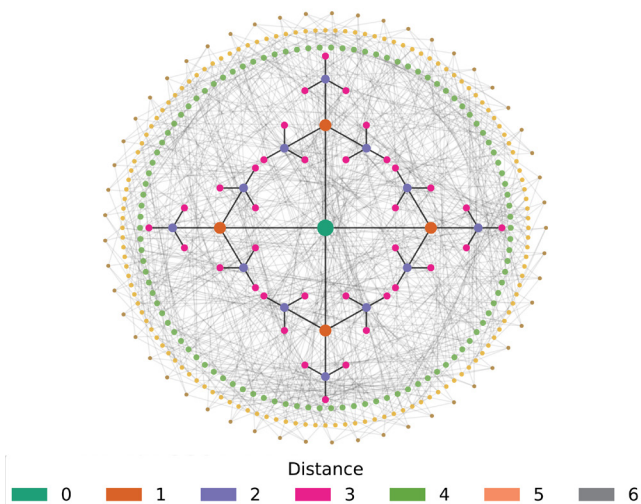


FIGURE 4. In this depiction of the connections between switches for a 336-switch SpectralFly topology with four intra-switch connections per switch, the switches are color coded by distance from a central switch, highlighting the tree-like neighborhood of the switch.

torus mesh. We consider a small, sparse SpectralFly network on 120 nodes and 240 links. To ensure a fair comparison, we optimally select^b the parameters of a DragonFly topology on exactly the same number of nodes and edges, and a two-dimensional torus mesh on 121 nodes and 242 edges. This near-exact three-way match enables a size agnostic comparison: each network starts with the same number of nodes, links, and radix, but makes different design choices in assembly.

The three networks are visualized in [figure 6](#). Each row of [figure 6](#) plots the same network, but with one of three different structural properties emphasized: the tightest bottleneck, the network diameter, and link usage frequencies in random traffic. Each of these structural properties are fundamental for supercomputer design: bottleneckness measures congestion proneness, diameter is a proxy for worst-case latency, and link usage patterns impact link-contention.

Bottlenecks

The first column of [figure 6](#) presents a split of each topology into equal parts which minimizes the number of edges crossing (in red), as found by METIS software [16]. For SpectralFly, Dragonfly, and torus, this yields 40, 31, and 26 links crossing,

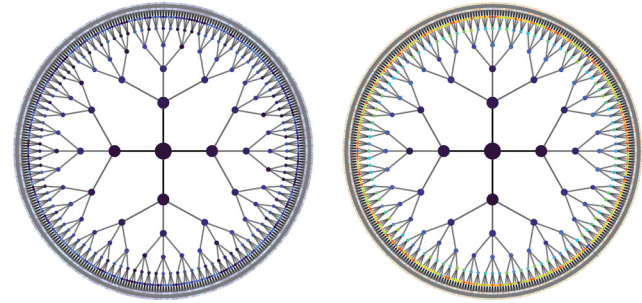


FIGURE 5. In this comparison of an infinite tree for a two-dimensional torus topology (left) and a SpectralFly topology with parameters (3,7) (right), the vertices of both are colored using the same equally spaced gradient. The vertex color corresponds to the order the vertices are discovered in the process; earlier vertices are colored blue and later vertices are colored red. As we can see, the torus topology is significantly less colorful than the SpectralFly topology, indicating that the SpectralFly topology has significantly better expansion properties.

respectively. In practice, this means that when there are many messages, we would expect the communication delays to be about 24% smaller as compared to DragonFly, and 35% smaller as compared to the torus.

Diameter

[Figure 6](#)'s second column visualizes paths linking a source-destination pair furthest from each other in the network—the length of which is known as the network diameter. The k -th ring of vertices from the leftmost contains all those that can be reached from that vertex in k hops. Small diameters ensure any vertex can be reached quickly from any other. In this case, both SpectralFly and DragonFly have an identical diameter of 6, while the torus has a diameter of 10.

Link loading

We simulate unstructured traffic on each network by randomly selecting 5,000 source-destination pairs in each network, and then routing via a minimal path. In the case that there are multiple such minimal paths, we select one at random. For each link in the network, we count the number of times it was traversed. [Figure 6](#) presents the distribution of these link usage counts. For SpectralFly, this distribution is highly symmetric and tightly concentrated, reflecting that edges are evenly spread across the network.

b. In particular, we generate a DragonFly topology with height $h=1$, $g=24$, groups of size $a=4$. We optimally allocate the intergroup edges as suggested by Teh, Wilke, Bergman, and Rumley [15].

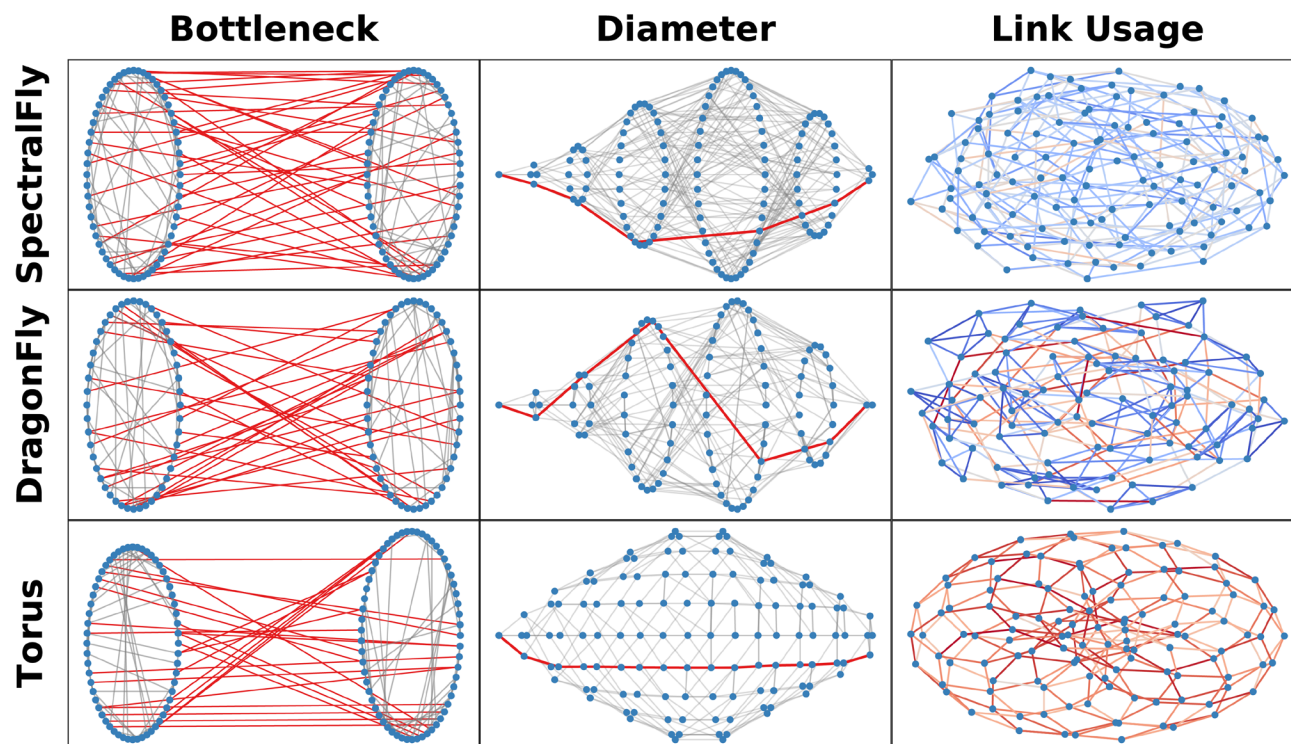


FIGURE 6. These graph visualizations emphasize different structural properties for three similarly-sized SpectralFly, DragonFly, and torus instances, each on about 120 nodes and 240 edges. The first column emphasizes the expansion, the second column emphasizes the diameter, and the third column emphasizes the prevalence of edges on shortest paths.

DragonFly, on the other hand, is the opposite and has a long tail: some edges are used heavily while others are almost never used. Lastly, as with SpectralFly, the link usage counts for the torus are also tightly concentrated, but overall larger: due to the torus' larger diameter, paths linking vertices tend to be longer and edges get used more frequently, albeit evenly, across the network. These observations are also reflected in the network visualization: each edge is colored on a blue-to-red scale, according to its percentile within the observed link counts aggregated across all three networks. Accordingly, SpectralFly and the torus' edges are homogeneous in color; whereas, those in Dragonfly run the gamut.

Conclusions and future work

As the workloads executed on current and future HPC systems evolve to include nontraditional workloads, such as data analytics and artificial intelligence (AI)/machine learning, so should the systems themselves. We argue that HPC systems should move away from

highly structured networks optimized for regular and large message communication to networks such as SpectralFly that expand and dynamically remove bottlenecks, and hence adapt to the irregular and unpredictable nature of the workloads. This move however would be onto a road less traveled, and as such, will require strong evidence that it can efficiently support emerging application domains before industry will commit to investing in it. At Pacific Northwest National Laboratory, we have developed and used several tools, based on MPI and partitioned global address space (PGAS), to analyze different network designs. The results indicate that SpectralFly networks are not only better at supporting irregular communication typical in data analytics and AI/machine learning, but that they might also outperform traditional networks when executing regular applications (unless they heavily rely on near-neighbor communication). In other terms, the SpectralFly network will let you sip your much deserved, end-of-the-day coffee at your favorite coffee shop without spending hours stuck in the car on the streets of Seattle downtown. 🌈

References

- [1] Adiga NR, Blumrich MA, Chen D, Coteus P, Gara A, Giampapa ME, Heidelberger P, Singh S, Steinmacher-Burow BD, Takken T, Tsao M, Vranas P. "Blue Gene/L torus interconnection network." *IBM Journal of Research and Development*. 2005;49(2-3):265–276.
- [2] Leiserson, CE. "Fat-trees: Universal networks for hardware-efficient supercomputing." *IEEE Transactions on Computers*. 1985;C-34(10):892–901. doi: 10.1109/TC.1985.6312192.
- [3] Kim J, Dally WJ, Scott S, Abts D. "Technology-Driven, Highly-Scalable Dragonfly Topology." *SIGARCH Computer Architecture News*. 2008;36(3):77–88. doi: 10.1145/1394608.1382129.
- [4] Chung FRK. "On concentrators, superconcentrators, generalizers, and nonblocking networks." *The Bell System Technical Journal*. 1979;58(8):1765–1777. doi: 10.1002/j.1538-7305.1979.tb02972.x.
- [5] Pippenger N. "Superconcentrators." *SIAM Journal on Computing*. 1977;6(2):298–304. doi: 10.1137/0206022.
- [6] Hoory S, Linial N, Wigderson A. "Expander graphs and their applications." *Bulletin of the American Mathematical Society*. 2006;43(4):439–561. doi: 10.1090/S0273-0979-06-01126-8.
- [7] Singla A, Hong C, Popa L, Godfrey PB. "Jellyfish: Networking data centers randomly." In: *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*; 2012. Available at: <https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final82.pdf>.
- [8] Valadarsky A, Shahaf G, Dinitz M, Schapira M. "Xpander: Towards optimal-performance datacenters." In: *Proceedings of the 12th International Conference on Emerging Networking EXperiments and Technologies*; 2016, pp. 205–219. doi: 10.1145/2999572.2999580.
- [9] Aksoy SG, Bruillard P, Young SJ, Raugas M. "Ramanujan graphs and the spectral gap of supercomputing topologies." *The Journal of Supercomputing*. 2021;77(2):1177–1213. doi: 10.1007/s11227-020-03291-1.
- [10] Alon N. "Eigenvalues and expanders." *Combinatorica*. 1986;6(2):83–96. doi: 10.1007/BF02579166.
- [11] Young S, Aksoy S, Firoz J, Gioiosa R, Hagge T, Kempton M, Escobedo J, Raugas M. "SpectralFly: Ramanujan graphs as flexible and efficient interconnection networks." In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Lyon, France, 2022 pp. 1040–1050. doi: 10.1109/IPDPS53621.2022.00105.
- [12] Lubotzky A, Phillips R, Sarnak P. "Ramanujan graphs." *Combinatorica*. 1988;8(3):261–277. doi: 10.1007/BF02126799.
- [13] Margulis GA. "Explicit group-theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators." *Problemy Peredachi Informatsii*. 1988;24(1):51–60.
- [14] Ramanujan S. "On certain arithmetical functions." *Transactions of the Cambridge Philosophical Society*. 1916;22(9):159–184.
- [15] Teh MY, Wilke JJ, Bergman K, Rumley S. "Design space exploration of the dragonfly topology." In: Kunkel J, Yokota R, Taufer M, Shalf J (Eds), *High Performance Computing. ISC High Performance 2017. Lecture Notes in Computer Science (LNTCS)*, vol 10524. Springer, Cham. Available at: https://doi.org/10.1007/978-3-319-67630-2_5.
- [16] Karypis G, Kumar V. "A fast and high quality multilevel scheme for partitioning irregular graphs." *SIAM Journal on Scientific Computing*. 1998;20(1):359–392. doi: 10.1137/S1064827595287997.