TorusVisND: Unraveling High-Dimensional Torus Networks for Network Traffic Visualizations

Shenghui Cheng[1], Pradipta De[1], Shaofeng H.-C. Jiang[2], and Klaus Mueller[1]

[1]Computer Science Department, Stony Brook University and SUNY Korea
[2]Computer Science Department, The University of Hong Kong

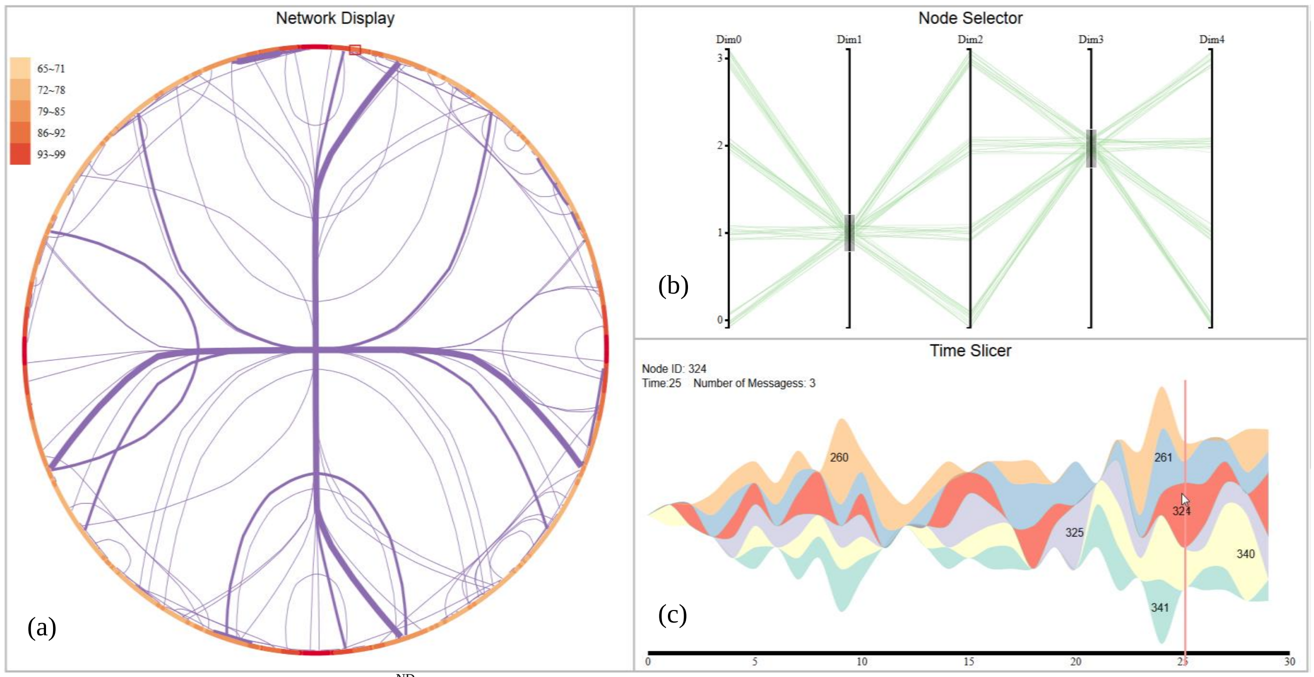


Figure 1: Our visualization framework TorusVisND and its three components: (a) network display (b) node selector and (c) time slicer. The network display currently shows the traffic across the nodes chosen in the node selector when time=25 as selected in the time slicer. The time slicer shows the message flow across the nodes chosen in the red rectangle.

***Abstract*— Torus networks are widely used in supercomputing. However, due to their complex topology and their large number of nodes, it is difficult for analysts to perceive the messages flow in these networks. We propose a visualization framework called TorusVisND that uses modern information visualization techniques to allow analysts to see the network and its communication patterns in a single display and control the amount of information shown via filtering in the temporal and the topology domains. For this purpose we provide three cooperating visual interfaces. The main interface is the network display. It uses two alternate graph numbering schemes – a sequential curve and a Hilbert curve – to unravel the 5D torus network into a single string of nodes. We then arrange these nodes onto a circle and add the communication links as line bundles in the circle interior. A node selector based on parallel coordinates and a time slicer based on ThemeRiver help users focus on certain processor groups and time slices in the network display. We demonstrate our approach via a small use case.**

***Keywords*—Torus network; visualization; topology**

I. Introduction

High performance computing has become an indispensable tool for the simulation of phenomena infeasible to enact in real physical experiments, ranging from climate to nuclear to astrophysics and even economics. The size and complexity of these problems has been steadily increasing, and so has the size and complexity of the computational hardware. Supercomputers can now have tens and even hundreds of thousands of highly interconnected compute nodes and this trend has no end in sight. Given these complex and massive network topologies, recognizing and trouble-shooting irregularities in inter-processor communications and data traffic can be exceedingly challenging. One of the most basic such challenges is how to overview and browse all nodes and their interconnections in an effective way. While this is simple for a 2D or even 3D mesh configuration, it becomes quite involved with a 3D torus network, and is currently unthinkable for the new state-of-the-art 5D torus networks, such as the IBM Blue Gene/Q. In this paper, we present a fledgling framework that is designed to fulfill these imperative needs.

A torus network is a type of interconnect network in which nodes can connect to their neighbor nodes in form of a mesh. The last node in each dimension can connect to the first node as well. In this case, the torus network becomes symmetric. The torus network has high dimensionality – 3D, 5D or even 6D – and a topology which is difficult for users to understand. Moreover, because of the high dimensionality, it usually consists of a large number of nodes and the complex communication patterns among the nodes can easily lead to confusion. To allow users to better understand the torus network for the purpose of real-time monitoring, timing analysis, or debugging, an effective visual interface can be of great help.

For this visual interface, our principle design goal was to provide a single 2D view onto the network (as opposed to a small multiples visualization composed of many projections) and allow users to manage the visualization of the possibly massive number of nodes, links, and time slices via a set of complementary interactive widgets. In network traffic analysis the proximity of nodes is a major determinant of the emerging patterns in processor inter-communication. Hence we seek a 2D mapping that can (1) emphasize local node neighborhoods connected in the high-dimensional torus topology, and (2) provide sufficient room to visualize the node interconnections.

A 2D space embedding via techniques like Multidimensional Scaling (MDS) [14] fulfills the first goal but leaves the display too crowded to meet the second goal. A better solution is to create a 1D embedding of the network and draw the links into the second dimension. Further, for understandability of the embedding it is desirable to devise a systematic traversal of the node space. Non-linear MDS is an optimization technique which makes heavy use of randomization and will not fulfill this goal. A sequential scan-line ordering along hyper-rows leads to a systematic ordering but has severe discontinuities at the end of the hyper-rows. These discontinuities could be prevented by incrementing the node indices in appropriate ways, but essentially such a scheme results in the first stage of the recursive fractal-like pattern generated by a space-filling curve, such as Hilbert or Peano.

Our paper studies the use of linear embedding techniques as a way to visualize high-dimensional torus networks. Specifically we consider sequential and space filling curves. Our complete design augments these 1D embeddings with the node-connective links and provides various interaction widgets to select interesting node neighborhoods, communication paths, and time slices. We describe the details of our system in Sections 3, and 4 after presenting some related work in Section 2. Conclusions and future work end the paper in Section 5.

II. Related Work

A recent STAR report [13] provides a general survey of the techniques proposed to visualize various aspects associated with high performance computing systems, such as profiles, traces, call graphs, I/O, software, memory, and others. Our technique makes use of a radial organization of the linear node embedding. Radial layouts have been used before in performance visualization, but to the best of our knowledge not for the visualization of the torus network itself. Choudhury and Rosen [2] use a radial layout for the display of memory hierarchies, where the outer ring represented the main memory, and inner arcs coded various levels of cache with the processor at the center. Cornelissen et al. [4] use radial layouts to visualize serial traces. They organize the methods on the outer circle and connected them with edges cutting across the inner area. Similar to our framework, they also make use of edge bundling [9] to prevent clutter in the interior region. A general overview on radial layouts is provided by Draper et al. [5].

Closest to our mission – network traffic visualization – is the work by Landge et al. [12] which uses the Boxfish system [11] to visualize a 3D torus network by a set of occlusion-free 2D projections. These projections are easy to read once the concept of the visual encoding is understood, but the underlying projection method does not scale well to torus networks of higher dimensionality, such as the 5D torus network our technique can visualize. Likewise, the torus visualization method described by Bjørnstad [1] is also limited to the 3D graph projection.

A torus network is a high-dimensional structure. There are numerous visualization methods to deal with high-dimensional data. In parallel coordinates [10] the attributes define the vertical axes while the samples form patterns of polylines. Likewise, in Radviz [8] and in the Generalized Barycentric Coordinate plot [15] the attributes constitute the vertices of a regular sided polygon and the samples form patterns in its interior. In all of these modalities the axes or vertices, respectively, are placed in regular and predefined ways and do not create diagnostic patterns on their own. A biplot [6] or a dynamic scatterplot [17], on the other hand, is more descriptive since the attribute axes projecting into the sample distribution's PCA basis do provide some insight about their similarity in terms of the data distribution. We have already mentioned general low-dimensional space embedding techniques, such as MDS [14], linear discriminant analysis (LDA) used by Choo et al. [3], and others. These types of visualization methods, however, lose the topology information, that is, one can no longer see which points are direct connective neighbors and which ones are not. Conversely, our framework maps the nodes onto a line first, and then folds them into a circle to facilitate the torus connectivity.

III. Overview

Our TorusVisND framework consists of three linked components: the *network display*, the *node selector* and the *time slicer*. These three components are shown in Figure 1.

The network display (Figure 1a) shows the network's topology by arranging all processor nodes onto a circle in an order determined by the linear embedding strategy (sequential or space-filling curve). The links between the nodes are visualized as edge bundles to prevent clutter. The figure here only shows the processor links active in a certain time slice.

The node selector (Figure 1b) is a parallel coordinate display with each axis mapped to a torus network dimension (here 5). It allows users to select and filter certain processor address ranges for display in the other two interfaces, or to visualize the active processors as polylines.

The time slicer (Figure 1c) is a standard ThemeRiver display where each stream is mapped to one processor (as selected with the node selector). It can show any node property over time – we currently display the number of messages sent and/or received by each processor within a certain time interval. Selecting a certain time slice updates the link visualization in the network display correspondingly.

IV. THE NETWORK DISPLAY

The network display is the core component of our framework. In the following we first present some relevant theory and then describe our implementation.

A. *Nodes and Channels*

A network typically consists of nodes and channels. The topology of such a network can be described by an undirected graph. $G = (V, E)$ in which the vertices V are the nodes of the network and the edges E are the links or channels. The torus network is a type of network, but it has specific conditions for V and E. In the definitions given below we follow the descriptions of Nesson et al. [18].

Nodes: Suppose the n-dimensional torus network consists of K_i nodes in each dimension, where $K_i \geq 2,\ 1 \leq i \leq n$. In total, there are then $N = \prod_{i=1}^{n} K_i$ nodes. Each node in the torus has unique coordinates – the node *offset* $(x_1, x_2, \ldots, x_n)$, where $0 \leq x_i \leq K_i - 1,\ 1 \leq i \leq n$. The offsets of all nodes can be expressed in the node coordinate matrix C:

$$C = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{Nn} \end{bmatrix}$$

In a practical application, one typically chooses $K_i = 2^p$ $(i = 1,2,\ldots,n)$ where p is an integer. In this paper we only deal with these cases. Our test example is the 5-D torus network with 4 nodes in each dimension, yielding 1,024 nodes in total.

Channels: In dimension j, $1 \leq j \leq n$, the connectivity for node $X(x_1, \ldots, x_j, \ldots, x_n)$ is

$$X \rightarrow \begin{cases} \left(x_1, \ldots, (x_j - 1) mod\ K_j, \ldots, x_n\right), \\ \left(x_1, \ldots, (x_j + 1) mod\ K_j, \ldots, x_n\right). \end{cases}$$

We can find a node and its neighbors by means of connectivity. Each node has 2 neighbor nodes in each dimension and $2n$ neighbor nodes in total. In other words, the degree of a node in the n-dimensional torus is $2n$. Since the last node can connect to the first node when they are in the same dimension, it is easy to define the distance between two nodes. Let $Dist(*)$ be the distance between node X $(x_1, \ldots, x_i, \ldots, x_n)$ and node Y $(y_1, \ldots, y_i, \ldots, y_n)$ in the torus network, then:

$$Dist(X,Y) = \sum_{i=1}^{n} min(|y_i - x_i|,\ K_i - |y_i - x_i|).$$

B. *Torus Network Linearization via Node Ordering*

High-dimensional spaces are naturally difficult to comprehend. This is true for general data spaces and also for the network torus when n>2. Our aim is therefore to obtain a mapping that embeds the torus network into a lower-dimensional representation that can be easily displayed and appreciated. One such mapping is an arrangement in which the torus nodes are ordered along a line. When such an operation is executed on a graph, it is called a vertex *numbering* or *indexing* [16].

Our framework extends the technique of node ordering from graphs to torus networks which poses some special challenges. Since the torus network is symmetric, if we layout the nodes along a line, the start and end nodes will break the symmetry rule. A solution to the problem is to simply tie the two ends of the line together and form the circle shown in Figure 1a. As an added benefit, this also makes the arrangement more compact and allows the links to be drawn in the circle's interior (see below).

An important metric for these types of numberings is *locality*. Specifically for our application, we desire that nodes that are close in the ordering are also close in the torus network. But can we wish for the converse as well, that is, can we ensure that nodes that are closely connected in the torus network are also mapped to nearby locations in the optimal numbering? It turns out that only the former can be fulfilled (see [16] and also others). This is not surprising because such a numbering is essentially a dimension reduction which is often a lossy undertaking. A direct implication of this finding is that there will be pairs of nodes that might be far apart in the ordering but closely connected in the torus network.

The amount of locality that can be achieved depends on the type of node ordering, or *curve* across the high-dimensional domain. The locality can be quantified as follows. Suppose the index curve is C, and $d(*)$ is the distance function according to the curve index. Then the locality L can be measured as:

$$L = \sum_{i=1}^{N} w_{d(C(i),C(j))} Dist(V_i, V_j) \qquad (1)$$

where $\{j \in [1,N]; d(C(i),C(j)) \leq \text{k}\}$ and smaller values for L mean better locality. Essentially, we choose the k nodes nearest to the indexed node and then calculate their connective distances in the torus network. Typically, we set k=6. In addition, we also assign different weights w according to the index distance of these k points. As users pay more attention to the nearest points and less to points further apart, their weight of perceptual influence decreases. The weight function is defined as follows and plotted in Figure 2.

$$f(x) = e^{-(\frac{x}{2})^2}$$

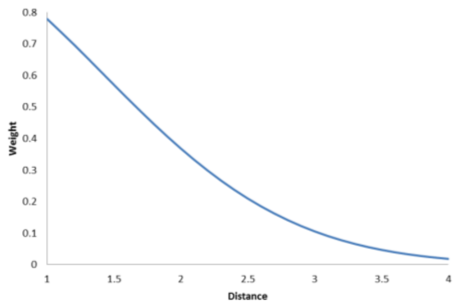


Figure 2: The weight function used to compute the locality metric.

C. *Node Ordering via Sequential Torus Nework Traversal*

The simplest curve for traversing the torus network is to go by offset. In this ordering there is a jump from the last point of one dimension to the first point of the next dimension. This means that two points adjacent in the sequence index might be far away in torus network space which breaks the locality. Using equation (1) the locality of this sequence layout is 1,822, which is relatively poor, as we will see shortly.

D. *Node ordering via Traversal with Space Filling Curves*

Superior locality can be achieved with curves that have a fractal character, such as the Hilbert space-filling curve [19][21]. A 2D Hilbert curve is shown in Figure 3. We can

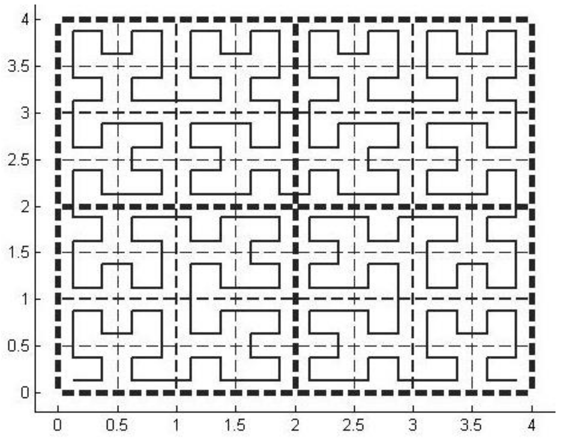


Figure 3: Hilbert curve in 2D

clearly see the self-similar (fractal) and hierarchical character of the curve. There are 4 main quadrant blocks, each composed into 4 similar blocks again, and so on.

To traverse a 5D torus network we require a 5-dimensional Hilbert curve. Given 2^p nodes per dimension, p binary digits are needed to encode it. The index along the Hilbert curve can then be represented by an np-bit integer which can be decomposed into n bits of p binary digits each.

Using equation (1) we compute the locality of the Hilbert curve node ordering as 1414 – and improvement of 22.4% over the sequential ordering. As mentioned, the locality metric does not cover how many neighboring torus nodes may map to distant locations from a focus node. This is better assessed with the converse metric:

$$CL_i = \frac{\sum_{j=1}^{2n} d(V_i,V_j)}{2n} \left((V_i, V_j) \in E\right) \qquad (2)$$

Here, CL_i is the average distance of a node i and its direct neighbor nodes. Figure 4 colors the nodes according to CL_i, both for the sequential curve and for the Hilbert curve (darker reds denote higher *CL)*. The difference is not overly drastic, but we observe that the distribution around the circle seems smoother for the Hilbert curve. The inserts in each circle presents graphs that plot the local *CL* gradients and we observe that the Hilbert curve's *CL* transitions are indeed smoother, while the sequential curve has several large spikes.

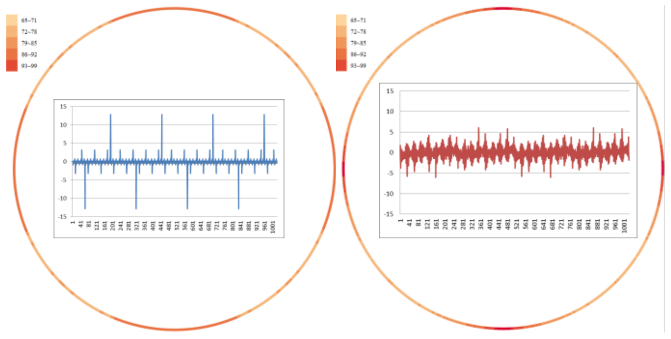

(a) (b)

Figure 4: Our two node ordering schemes compared: (a) sequential (b) Hilbert curve. The color of a node shows the average torus-space distance between it and its neighbors. The line charts in the center plot the *CL* changes (gradients) between two neighbor points on the circle.

D. Adding the node interconnections to the network layout

Having laid out the nodes optimally, in order to visualize the network traffic we need to show their direct interconnections. The radial layout is perfectly suited to depict interconnections of entities, as has been recognized also in other works in the area of performance visualization [4]. Essentially, we draw a line for each directly connected torus node. Figure 5 shows the outcome for both types of curves we have studied – sequential (a) and Hilbert (b). The visual clutter due to the massive number of direct connections (2*n*) is obvious.

But before we address the visual clutter we make another important observation. It appears that the sequential indexing leads to much empty space in the center, essentially wrapping around the circle's annulus. The Hilbert curve, on the other hand, makes better use of the circle interior. This may prove advantageous when it comes to readability, although the sequential scheme may be easier to understand conceptually. Testing these two hypotheses, as well as others, will be part of a set of user studies to be conducted in the future.

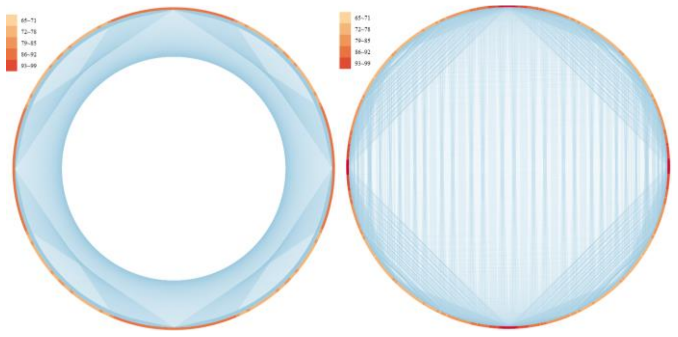

(a) (b)

Figure 5: Showing the direct node interconnections by interior lines (blue): (a) sequential curve indexing and (b) Hilbert curve indexing.

In order to reduce the visual clutter resulting from the crossing of many straight lines, we apply the popular edge bundling technique of Holten at al. [8]. It warps straight edges that both originate and end in similar areas into splines and bundles them together like cable trees. Edge bundling typically greatly reduces visual clutter, but as we observe in Figure 6, it has only little effect on the sequential curve layout (a) while it provides much better readability for the Hilbert curve (b).

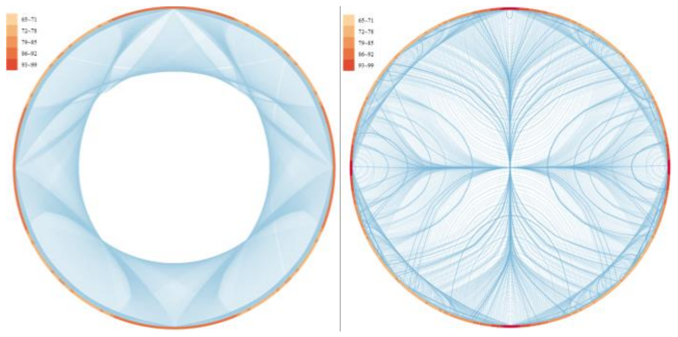

(a) (b)

Figure 6: The effect of edge bundling: (a) sequential curve indexing and (b) Hilbert curve indexing.

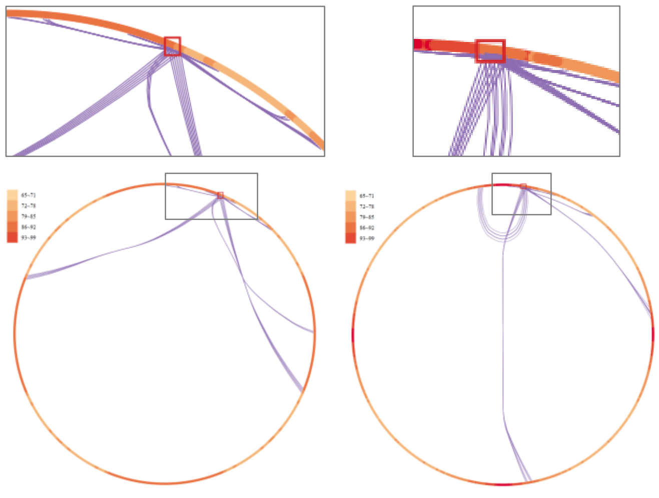

(a) (b)

Figure 7: Visualizing the connections of a small group of (root) processors with their directly connected neighborhoods: (a) the sequential indexing scheme and (b) the Hilbert curve indexing scheme. For each panel, the bottom figure shows the overall distribution and the top figure shows the zoomed-in area around the root processors.

E. *Contrasting the visual signatures of the ordering schemes*

Zooming into a small neighborhood of nodes on the circular layout gives insight into the visual signatures of the two ordering techniques we studied. This is shown in Figure 7 where we examine the first connections of a set of 6 processors for a 5D torus network. Here, Figure 7a shows the pattern of the sequential curve indexing scheme and Figure 7b shows the patterns of the Hilbert curve indexing scheme. In each figure, the bottom shows the overall distribution and the top shows the zoomed-in area around the root processors.

The sequential scheme clearly expresses the torus coordinate distances in the indexing distances. In the zoom-in we can see the communications within the primary coordinate as two short bundles of lines going in the two opposite directions, and we can also see the communications with processors that vary in the 2nd and 3rd most significant coordinate. These visualize as four line bundles reaching out a bit further. The final two, least significant coordinates give rise to the line bundles reaching far out.

The Hilbert curve scheme exhibits a rather different pattern. The first observation we make is that the connections occur, at least on average, on a more local level. There is only one line bundle that goes to the opposite of the circle since these processors happen to reside in a different hyper-cube at the top level of the fractal hierarchy. The other bundles are due to processors also residing in different hyper-cubes than the root processors but at decreasing levels in the fractal hierarchy.

For the remainder of this paper we will focus on node orderings generated with the Hilbert curve numbering scheme. While is too early to pass judgment on which of the two schemes (or even another) is more intuitive to domain users – we would require a user study and real data for this – we believe, at least for now, that the locality behavior of the Hilbert scheme, and the fact that it makes better use of the circle interior space, makes it the preferable method.

V. The Node Selection Interface

A realistic torus network can have tens and even hundreds of thousands of highly interconnected compute nodes. This will amount to an extremely crowded network display, as has been already demonstrated in Figure 6. In the presence of large data and attributes, selection, filtering and brushing are effective techniques to control the deluge. And so, we also require an effective interface that allows analysts to focus on interesting subsets of torus network processors for the purpose of studying their communication patterns in the network display.

The coordinates of the processors constitute a familiar organizational encoding for a large parallel computer such as the torus network, and so it is meaningful to build a selection interface around this organizational structure. With this in mind, we have implemented a parallel coordinate interface [10] which assigns each node coordinate (dimension) to one of the parallel axes, in increasing order from left to right. Thus, a 5D torus network gives rise to a parallel coordinate display with five parallel axes. In this visualization, a specific processor is expressed as a single *polyline* – a piecewise linear spline which connects the processor's coordinate positions (locations) on each of the parallel axes.

The parallel coordinate visualization of the 5D torus network is shown in Figure 8, plotting all processors and their respective polylines. The careful reader will notice that in this figure, the individual polylines appear slightly displaced and form narrow bands. This occurs because we assigned to each polyline a small random offset in each dimension. Had we not done this, it would have been difficult to visually trace individual polylines since they all would meet at a few discrete positions along a coordinate axis, as implied by their discrete integer coordinate values, such as 1, 2, 3, … The small offset does not change the coordinate values significantly, and the locations of the intersection points are still around the true coordinate values, but the overlap is reduced and the density of lines with a specific coordinate value can be easily discerned.

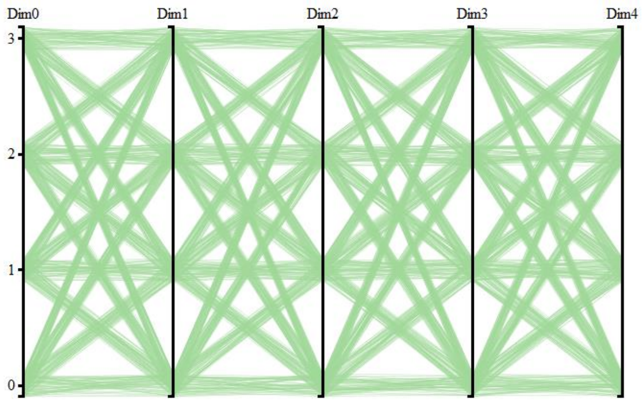


Figure 8: the parallel coordinate-based node selction interface

A. *Node selection*

We can now easily select a processor by simply clicking on a polyline and see it and its communication links highlighted in the network display, with the other links shown as background to provide context. This is shown in Figure 9 where the red line is the selected processor with coordinate (0, 1, 2, 2, 0). Vice

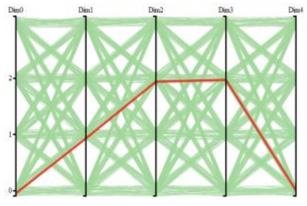
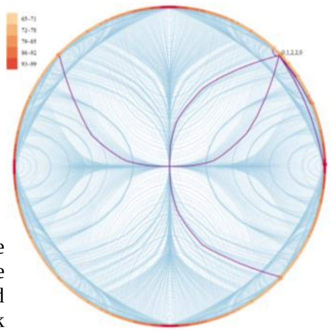

(a)

Figure 9: Selecting a node with the parallel coordinate display: (a) the selected processor highlighted as a red polyline, (b) the coupled network display with the selected node and its direct communication inks highlighted – the other torus links are shown in the background for context.


(b)

versa we can also pick a node or communication link in the network display and see these processors highlighted in the node selector.

B. *Node filtering and bracketing*

It can also often be useful to select a group of processors and see their intercommunication patters. We did this in our earlier example depicted in Figure 7. This can be achieved by filtering and bracketing operations in the node selector's parallel coordinate interface. Figure 10 shows an example where we selected processors with specific coordinate values in the 3rd and 5th dimensions. Their links are then highlighted in the associated node display.

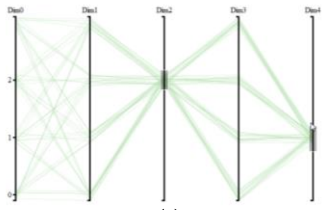
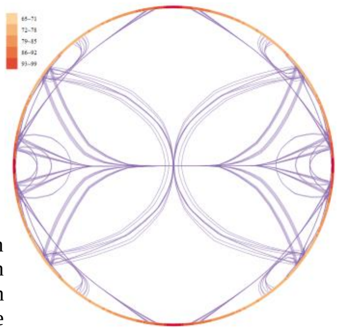

(a)

Figure 10: Filtering and bracketing with the node selector. (a) Only nodes with specific values in the third and fifth coordinates are selected (b) The corresponding nodes in and their communication links in the network display.


(b)

There might be communication links that have more weight than others. The weight could be due to many factors, such as importance, number of messages in a certain time interval, or the number of processors in the selected set using it (one or two when only directly connected processors are considered – more when also processors are considered that have used the link in a wider-range path). A relatively straightforward way to show this weight is by using different line strength or opacity. Figure 11 presents an example.

VI. Use Case – Traffic Visualization

To show a first use case of our framework, we simulated a simple network traffic scenario. In this simulation, we assume that (1) all nodes can execute the code correctly and without fail, and (2) the bandwidth of the channels is sufficient to allow all messages to pass through without jam.

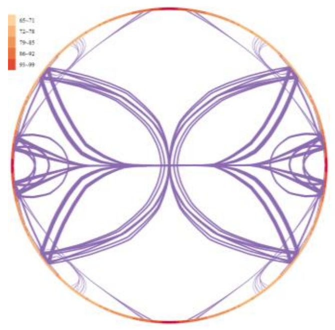


Figure 11: Highlighting important links by higher opacity.

Pseudo code of our simulation algorithm is presented Figure 12. The time generation is similar to a wake up – the processor starts to become active and sends/receives messages. We generate the wakeup time t_i for a node i at random after which it sends a message to one randomly selected neighbor. All nodes are set to wake up before half of the full simulation time T has expired (T=30 minutes). When node i sends a message at time t_i its neighbor receives it at $t_i + 1$, and sends it to its randomly chosen neighbor who receives it at $t_i + 2$, and so on. The process continues until T has been reached.

```
Algorithm 1: Traffic Simulation Algorithm
1. Initialization
    For i=1:N
      For j=1:T
        Node[i][j]={time: i, msg:0 };
2. Wake up time generation:
        For i=1:N
          T[i]=(random()*N/2).floor();
3. Message generation:
        For i=1:N
          Node[i][T[i]].msg++;
            oC=i;     //old choice
            nC=i;     //new choice
              For i=T(i)+1:T
                nC=Node[oC].neighbor[(random()*2n).floor()];
                Node[nC][i].msg++;
                oC=nC;
```

Figure 12: Our network traffic simulation algorithm

A. *Track a message across the network*

We first use the network display to track a certain message across a series of nodes. In Figure 13, the color tone of the path indicates time – lighter mean older and darker is more recent.

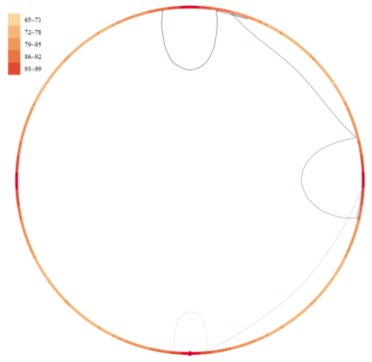
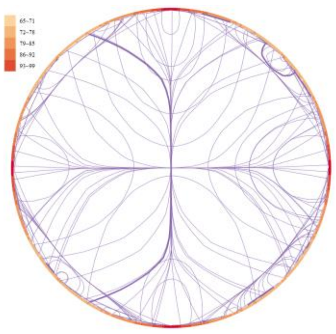


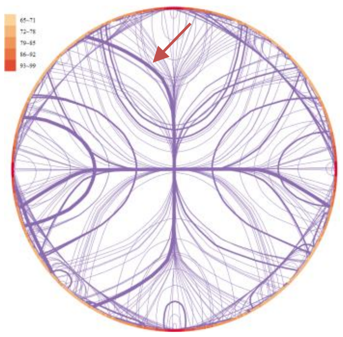
Figure 13: Tracking a single message across the network. The color tone of the path indicates time – lighter means older and darker means more recent.

B. *Visualize overall network traffic*

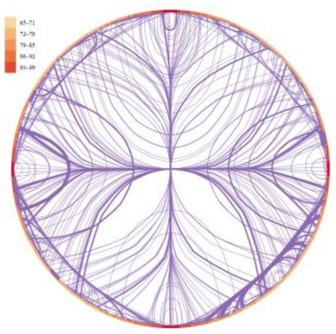
Our next example visualizes overall network traffic. Figures 14a-c show three snapshots along time, each of a certain time slice. In these visualizations, edges are only shown when there exists a transmission and edge opacity is mapped to the number of messages sent through the link within this time slice. We can clearly see that as time goes on there are more messages in the network (since more nodes have woken up). But we can also observe that at *t*=10 (Figure 14b) the link pointed to with red arrow seems rather busy, but is less busy at *t*-15 (Figure 14c). Finally, Figure 14d shows the network when only a subset of processors are considered – those with a the second coordinate set to 1 and fourth set to 2. We can clearly observe that one particular links is much busier than the others which can point to possible future link contentions.



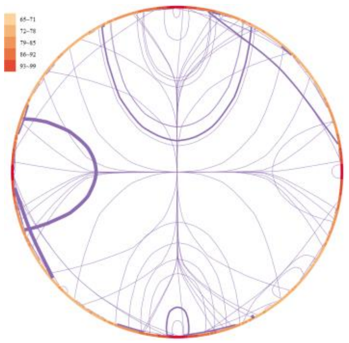
(a) *t* = 5



(a) *t* = 10



(c) *t* = 15



(d) *t* = 10 filtered with selection below

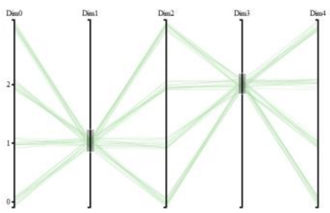


Figure 14: Network traffic state at various time slices. Heavier colored edges mean more traffic. We can observe the change in network traffic over time, and we use the node selection interface to focus on a certain set of processors.

C. *Select and visualize network activity for a time slice*

Next we show how our system allows analysts to examine a certain time period, and optionally a specific processor or processor group as specified with the node selector.Our system uses ThemeRiver [7] to visualize the time-series data of the nodes and/or links and empowers users to make comparisons among them. ThemeRiver creates an axis-centered, stacked stream graph of the set of time series data where the height of an individual stream is proportional to the data value.

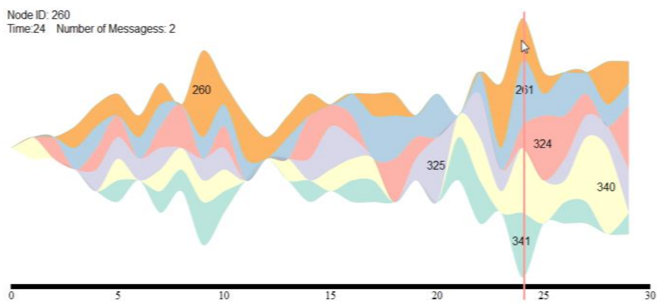
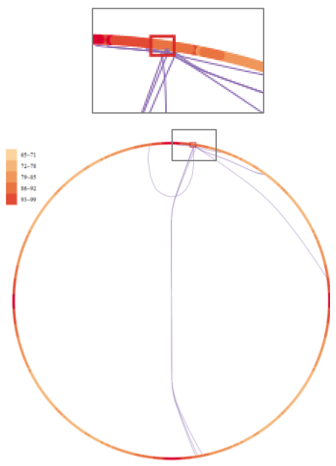


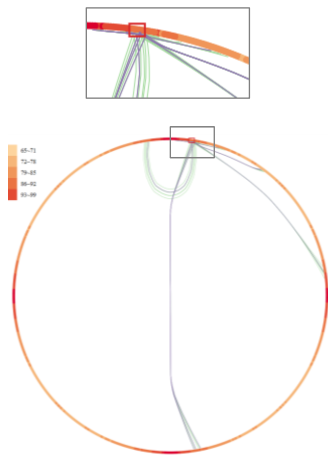
Figure 15: Our time slicer uses the ThemeRiver paradigm to visualize the messages flow across a set of selected nodes. The black line at the bottom is the time axis. The vertical pink line is a user-selected time slice. The labels of the different layers are the node IDs. Upon moving the mouse overs a specific layer, the system displays the node ID, current time and the number of messages of the corresponding node.

Figure 15 shows a ThemeRiver display for the number of messages going through the nodes directly connected to the root node marked in Figure 7. We have colored the layers of ThemeRiver with different colors and have used the node index to order them. We can make a number of interesting observations. First, every node has periods in which it does not receive or send a message. For example, node 260 for example sends no message when *t*=1, 2, 16, 17, 19, 20, 21, 27 and 28. It was waiting for the message at these time periods. Second, we can also tell when a node has a high number of messages and seems to be busy. For node 260, this occurs when *t*= 9, 28, etc. Finally, we can also observe the overall message flow across these nodes by gauging the entire width of the stream. Here we observe that when *t*=12, the number of messages from all nodes are smallest and when *t*=24, they are largest.

In our last example we utilize the time slicer to pick a specific time slice (*t*=24), and we use the node selector to pick a set of root nodes (those that were already used in the previous example). The slice at *t*=24 (see Figure 15) is the busiest and so analysts might be interested in seeing where the traffic comes from. Upon these selections the network display shows the (busy) links involved in these messages in blue (Figure 16a).



(a)



(b)

Figure 16: Network traffic analysis. Upon selecting a time slice and a set of processor nodes the network display can show (a) the set of nodes that are involved in the messages (links colored blue) and (b) which one are not (links colored green).

Since it is insightful to also see the links not involved in these communications, our system provides the option to color these (silent) links in green (Figure 16b).

VII. Conclusions

The main premise that guided our development of TorusVisND was to give torus network performance analysts a framework that can provide a single view onto the network, as opposed to an array of projective views. We believe that such an approach scales better with the increasing dimensionality and magnitude of these networks, as they seek to grow performance, bandwidth, and communication speed.

In our effort we made use of the concept of graph numbering. We experimented with two such schemes – sequential curve indexing and Hilbert curve indexing – and studied them via a set of simple use cases, tasks, and scenarios. We found that each method has advantages and disadvantages, as pertaining to readability and locality. Exposing the system to real network analysis and running it with real performance data will probably bring more clarity to these issues.

Our framework follows one of the classic paradigms of information visualization – overview, filter, and detail on demand. Known as the "Visual Information Seeking Mantra" [20] it puts forward a mindset where users are in the loop, steering the data exploration process via operations like selection, filtering, and brushing. We achieve these operations via our node selector interface based on the method of parallel coordinates and with our time slicer based on the ThemeRiver paradigm. Both can be used for selection and filtering, but also for the visualization of network performance.

Filtering and selection help users manage large and complex data and configurations, and we hope to have demonstrated that this has also great potential for torus network performance analysis. Future work will aim to make our interface more scalable. In particular we would like to introduce multi-resolution capabilities into the network display to allow it to handle larger numbers of network nodes, and we would also like to introduce multi-perspective lenses to the network display interior to allow users to zoom into multiple areas of interest. Finally, we would like to work with domain experts and real data to truly optimize our framework and system.

Acknowledgments

This research was partially supported by NSF grant IIS 1117132 and the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the "IT Consilience Creative Program (ITCCP)" (NIPA-2013-H0203-13-1001) supervised by NIPA (National IT Industry Promotion Agency).

References

[1] J. Bjørnstad, *3D Visualisation of Network Topology, Routing, Path Distribution and Network Data in Simulated InfiniBand Clusters*. Masters thesis. Department of Informatics. University of Oslo, 2011.

[2] A. Choudhury and P. Rosen, "Abstract visualization of runtime memory behavior," *Proc, IEEE Workshop on Visualizing Software for Understanding and Analysis*, pp. 1-8, 2011.

[3] J. Choo, S. Bohn, H. Park. "Two-stage framework for visualization of clustered high dimensional data," *Proc. IEEE VAST*, pp. 67-74, 2009.

[4] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. Van Wijk, A. Van Deursen. "Understanding execution traces using massive sequence and circular bundle views." *IEEE Program Comprehension*, pp. 49-58. 2007.

[5] G. Draper, Y. Livnat, and R, Riesenfeld. "A survey of radial methods for information visualization." *IEEE Trans. on Visualization and Computer Graphics*, 15(5): 759-776, 2009.

[6] K. Gabriel, "The biplot graphic display of matrices with application to principal component analysis," *Biometrika*, 58(3): 453-467, 1997.

[7] S. Havre, E. Hetzler, L. Nowell: "ThemeRiver: Visualizing Theme Changes over Time," *Proc. IEEE InfoVis*, pp. 115-123, 2000.

[8] P. Hoffman, G. Grinstein, K. Marx, I. Grosse, E. Stanley, "DNA Visual and Analytic Data Mining", *IEEE Visualization*, pp. 437-441, 1997.

[9] D. Holten, J. van Wijk, "Force-directed edge bundling for graph visualization," *Computer Graphics Forum* 28(3): 983-990, 2009.

[10] A. Inselberg, B. Dimsdale, "Parallel Coordinates: A Tool for Visualizing Multi-Dimensional Geometry," *IEEE Visualization*, pp. 361-378, 1990.

[11] K. Isaacs, A. Landge, T. Gamblin, P. Bremer, V. Pascucci, and B. Hamann, "Exploring performance data with boxfish." *IEEE High Performance Computing, Networking, Storage and Analysis*, pp. 1380-1381. 2012.

[12] A. Landge, J. Levine, A. Bhatele, K. Isaacs, T. Gamblin, M. Schulz, S. Langer, P. Bremer, V. Pascucci, "Visualizing network traffic to understand the performance of massively parallel simulations," *IEEE Trans. Visualization and Computwe Graphics*, 18(12):2467-2476, 2012.

[13] K. Isaacs, A. Giménez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, T. Bremer, "State of the Art of Performance Visualization," *EuroVis 2014.*

[14] J. Kruskal. M. Wish, *Multidimensional Scaling* Sage Publications, 1977.

[15] M. Meyer, A. Barr, H. Lee, M. Desbrun, "Generalized Barycentric Coordinates on Irregular Polygons," *J. Graphics Tools*, 7(1):13-22, 2002.

[16] G. Mitchison and R. Durbin. Optimal numberings of an NxN array. *SIAM Journal on Discrete and Algebraic Methods*, 7(4):571–582, 1986.

[17] J. Nam, K. Mueller, "TripAdvisorN-D: A tourism-inspired high-dimensional space exploration framework with overview and detail," *IEEE Trans on Visualization and Computer Graphics*, 19(2): 291-305, 2013.

[18] T. Nesson and S. Johnsson, "ROMM routing on mesh and torus networks," *Proc. ACM Symp. on Parallel Algorithms and Architectures*, pp. 275-287, 1995.

[19] H. Sagan, *Space-Filling Curves*. New York: Springer-Verlag, 1004.

[20] B. Shneiderman, "The eyes have it: a task by data type taxonomy for information visualizations," *IEEE Symposium on Visual Languages*, pp. 336-343, 1996.

[21] D. Voorhies,"Space-filling curves and a measure of coherence," *Graphics Gems II*, pages, 26–30. Academic Press, 1991.