2014

CommGram: A New Visual Analytics Tool for Large Communication Trace Data

Jieting Wu
University of Nebraska-Lincoln, jwu@cse.unl.edu

Jianping Zeng
University of Nebraska-Lincoln, jizeng@cse.unl.edu

Hongfeng Yu
University of Nebraska-Lincoln, yu@cse.unl.edu

2014 First Workshop on Visual Performance Analysis
Pages: 28 - 35, DOI: 10.1109/VPA.2014.8

CommGram: A New Visual Analytics Tool for Large Communication Trace Data

Jieting Wu, Jianping Zeng, Hongfeng Yu
University of Nebraska Lincoln
Lincoln, Nebraska
Email: {jwu, jizeng, yu}@cse.unl.edu

Joseph P. Kenny
Sandia National Laboratories
Livermore, California
Email: jpkenny@sandia.gov

***Abstract*—The performance of massively parallel program is often impacted by the cost of communication across computing nodes. Analysis of communication patterns is critical for understanding and optimizing massively parallel programs. Visualization can help identify potential communication bottlenecks by displaying message trace data. However, the visual clutter and temporal incoherence problems are typically incurred in existing visualization tools for a considerable number of processors. In this paper, we present a new tool, named *CommGram*, which supports visual analysis of communication patterns for massive parallel MPI programs. With the benefit of MPI trace library DUMPI of SST, our framework builds hierarchical clustering trees for computational community domain, and takes advantage of graphical user interface (GUI) to convey communication patterns at different levels of detail. The effectiveness of our tool is demonstrated using large-scale parallel applications.**

I. Introduction

The advances of supercomputers enable researchers to improve the performance of their parallel programs to an unprecedented level. Performance analysis becomes ever more crucial to ensure the scalability of programs and maximize the efficiency of the machines. Due to the escalating disparity between data movement speed and compute speed [1], communication across computing nodes simply becomes a critical factor in overall performance, and researchers have particularly focused on collecting communication trace data to inspect large distributed programs for optimization. Substantial efforts have been made in the development of techniques and tools for detection and analysis of communication patterns. However, as the size and complexity of modern supercomputers increase, it becomes a rapidly severe problem that researchers can collect detailed communication trace data but lack scalable capabilities of data analysis to gain the insights.

To address the issue, visualization has received considerable attention from researchers because it offers a solution to analyzing large scale communication data. Many performance tools have been equipped with visualization capabilities that display and convey communication patterns by leveraging the powerful human visual perception [2], [3]. Despite the increasing demand for visualization, however, its usage has been limited because, for large graph data, it is easy to generate visual clutter and hard to perceive and track dynamic patterns.

In this work, we develop a novel tool, named *CommGram*, in support of visual analytics of large-scale communication data from massively parallel programs. We are inspired by the visual output of *ElectroCardioGram* in that the electrical activity of the heart is represented as a curve of wave over a period of time. Our tool can generate a similar output that uses a simple curve to represent the dynamics of communication patterns over time. More specifically, the amount of variance along the curve corresponds to the degree of changes of communication patterns: a flat section means that the communication pattern is largely constant, while a highly wavy section indicates that communication pattern is changing dramatically. This visual presentation provides a clear summarization of the time-varying communication events across many processors.

In addition, once a user detects an interesting pattern from the high-level overview, our tool allows the user to seamlessly zoom-in on a particular time interval and examine the detailed data exchanges across a particular set of processors. Our tool enables an interactive exploration of large communication data without visual clutter, and clearly shows the dynamic communication patterns in a coherent and scalable fashion.

It is non-trivial to develop such a visual analytics tool. We holistically address key design considerations from large communication data, to analysis requirements, to visualization properties. Our work makes the following contributions:

- We introduce a new visual representation that can clearly and coherently display time-varying graph based on the characterization of communication data.
- We present a method to construct a hierarchical abstract of a large communication trace data on computing nodes, mitigating the visual clutter issue.
- We develop a new user interface in cooperation with real-world trace tools, enabling an intuitive exploration of large communication event.

We demonstrate the effectiveness of our tool using the communication data generated from real-world large-scale applications. We show that the resulting visualizations can provide the informative depictions of communication and help simple monitoring and diagnoses of specific patterns in support of the performance characterization and optimization.

II. Related Work

A substantial amount of techniques and tools have been developed for parallel program analysis. The examples of early work on MPI programming include ParaProf [3], Umpire [4], and MPI-Check [5]. Recently, Hilbrich et al. [6] presented a tool named MUST based on their graph-based deadlock

978-1-4799-7058-2/14 $31.00 © 2014 IEEE
DOI 10.1109/VPA.2014.8

detection approach for MPI. Llort et al. [7] presented a framework that uses object tracking techniques for performance analysis. In particular, they clustered the performance information to convey evolution of a parallel application along multiple execution scenarios.

Besides these general techniques and tools, researchers also presented case studies to show the feasibility of performance improvement for large applications through detailed analysis. Sun et al. [8] used the Projections performance analysis tool to examine and optimize fine-grained communication in a biomolecular simulation application. They achieved a considerable improvement on a Cray XK6 system. Bhatele et al. [9] used a binary tree to visualize communication topology which facilitates a diagnosis of communication and workload distribution. They showed the performance improvement for a large AMR application on an IBM Blue Gene/P system.

Visualization is commonly used in the performance analysis. The representative visual designs include Gantt chart, Kiviat diagram, histogram, node-link diagram, and adjacency matrix. Many tools typically incorporate multiple types of visualizations. For example, ParaGraph [10], an early work on performance visualization, has included most of these visualizations. In particular, the tool represents a parallel system by a graph whose nodes represent processors, and whose arcs represent communication between processors. An animation of the graph shows the dynamic status of communication.

Jumpshot [11] used Gantt chart to provide an overview of system activities. Each MPI function call is represented as a chart segment with a different color. An arrow connecting the chart segments indicates a communication event and its corresponding processors. The similar Gantt chart based design has also been used in VAMPIR [2] and TAU [12]. In addition to Gantt chart, these two tools also employ the adjacency matrix representation to record the complete communication between every pair of nodes.

These conventional visualization techniques can intuitively present a communication graph. However, they all suffer from the scalability problem when the scale of communication increases: severe visual clutter can be incurred if the number of visual elements (such as nodes, arcs, and arrows) exceeds the number of available pixels for a display. To address this issue, Muelder et al. [13], [14] used high-precision alpha blending and opacity scaling techniques to overplot the visual representations of communication events and form larger patterns with low visual clutter. However, some detailed processor information can be lost. Sigovan et al. [15] extended the overplotting method and introduced a particle animation technique that can display every event in a trace data while maintain an overview of system activity. Sigovan et al. [16] also presented a visual network analysis method for the trace data from large parallel I/O systems. Landge et al. [17] introduced a method that combines the 2D and 3D views to visualize the network traffic of large-scale simulations on a supercomputer.

III. Our Approach

Exploring new scalable approaches to large-scale communication data analysis requires a holistic study. Traditional graph visualization techniques are not well suitable for new requirements imposed by the properties of communication

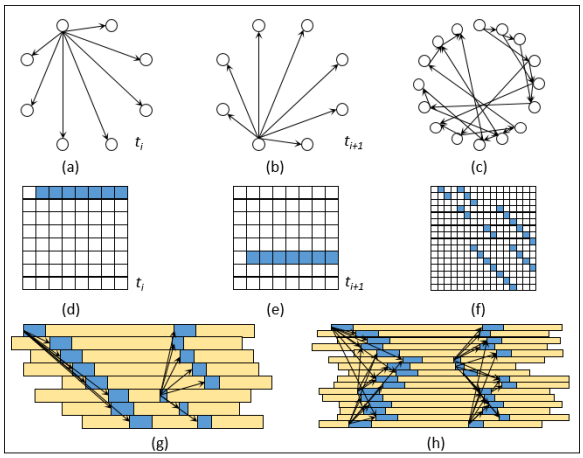

Fig. 1. Traditional visualization techniques. (a)-(c) show the node-link diagrams, (d)-(f) show the adjacency matrixes, and (g)-(h) show the Gantt charts for the same result, where (a)-(b) and (d)-(e) correspond to two consecutive time steps, and (c), (f) and (h) display the relatively large graphs. We can see that none of these techniques can resolve both the visual clutter and temporal incoherence problems.

of massively parallel programs. We introduce a novel graph transformation and utilize a corresponding hierarchical clustering technique to significantly increase the visual stability for analyzing large-scale time-varying communication graph data.

A. Intercepting Communication Trace Data

In this work, we study the communication patterns of MPI based parallel programs and use the DUMPI utility of the Structural Simulation Toolkit (SST) library [18] to collect communication trace data of MPI application calls. By wrapping the MPI functions, DUMPI can intercept the detailed communication functions from applications. The output of DUMPI can contain all the MPI function calls with the comprehensive information, such as the sender, the receiver, the message size, the time stamps, and so on. We can filter the DUMPI output to capture the certain peer-to-peer (P2P) communication functions (such as MPI_Send, MPI_Recv, MPI_ISend, and MPI_IRecv) and the collective functions (such as MPI_Gather and MPI_Reduce) in a simple logical order. These information allows us to construct a time-varying communication graph.

B. Revisiting Communication Graph Visualization

It is very difficult to use traditional graph visualization techniques to generate a visualization that can convey the temporal communication patterns in a coherent and clutter-free fashion. Figure 1 shows a set of examples. The node-link diagram can only show a simple graph clearly (Figure 1 (a) and (b)), and visual clutter can be easily generated when the number of nodes and edges increases (Figure 1 (c)). In addition, there is no coherence between the images of two consecutive time steps, and animating them (for example, Figure 1 (a) and (b)) cannot clearly show the transition between the patterns.

The adjacency matrix can avoid the visual clutter problem even for a large graph, as shown in Figure 1 (d), (e) and

(f). However, similar to the node-link diagram, the adjacency matrix cannot convey the change of communication patterns. Moreover, by comparing Figure 1 (a) and (d), we can clearly see that the communication pattern displayed by the adjacency matrix is less intuitive due to a lack of an explicit representation of the directional information.

The Gantt Chart can intuitively display the communication events using the arrows between the chart segments with respect to the processors. In addition, because the chart segments are sorted by time, different communication patterns can be distinguished over time, as shown in Figure 1 (g). However, with the increasing number of events, it is hard to avoid severe visual clutter from the Gantt chart, as shown in Figure 1 (h).

Researchers routinely perceive the visual clutter and/or temporal incoherence problems when they use visualization to analyze communication data collected from a large number of processors. To address these problems, we advocate that an appropriate design should be tailored to the characteristics of communication graph data, rather than directly adopt visualization techniques for general graph data.

C. Characterizing Communication Graph

Without loss of generality, in a communication graph, each node represents a processor and each directed edge represents a message sent from one processor (sender) to another processor (receiver). Compared to general graph data, a communication graph exhibits a set of unique characteristics.

First, because it is possible for a processor to be a sender and a receiver simultaneously during communication events, a communication graph can visually appear as a directed cyclic graph. However, the messages sent and received by a processor at a certain time are typically independent of each other, and thus the circles formed by the graph topology does not correspond to the flow path of data exchanged among the processors; that is, there is by no means a path along which a message sent by a processor will return to the processor again. Hence, a communication graph is typically modeled as directed acyclic graph (DAG) [19], or more specifically, communication directed acyclic graph (cDAG) [20]. This standard model, however, has been relatively overlooked in visualization studies: for example, with the popular node-link diagram method, a visualization may show the artifacts of circles in the communication graph and lead to a possible misunderstanding.

Second, a communication graph is not directly related to the network flow problem [21], in that the amount of messages received by a processor has no necessary relationship to the amount of messages sent by the processor. However, most advanced graph visualization techniques are developed based on network flows (e.g. [22], [23]), and thus are not suitable for communication graphs.

Our design foci has been accentuated to address the key usages of communication graphs. We note that a viable solution should facilitate an effective exploration of complex patterns, and enable a quick identification of potential communication bottlenecks, which is imperative for program optimization and performance improvement.

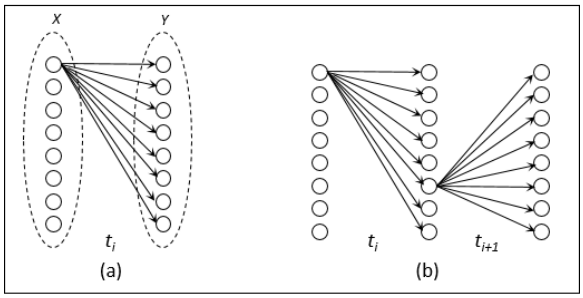

Fig. 2. (a) shows the bipartite graph equivalent to the communication graph in Figure 1 (a). We can concatenate the bipartite graphs over the time steps to represent the time-varying communication graph. (b) shows the result corresponding to the combination of Figure 1 (a) and (b).

D. Communication Graph Transformation

We propose a solution based on an equivalent transformation in that we use a sequence of bipartite graphs to represent a time-varying communication graph. Assume that we generate a time-varying communication graph G over a total of T time steps with n processors. We represent the processors as a set P of nodes, and their communications as the edges. We use $C_{t_i}(u,v)$ to represent an edge corresponding to a communication that a message is sent from a processor u to a processor v at the time step t_i:

$$C_{t_i}(u,v) = \{\overrightarrow{uv} \mid u \in P,\ v \in P,\ t_i \in T\} \tag{1}$$

The set of senders at t_i is represented as:

$$S_{t_i} = \{u \mid u \in P,\ t_i \in T\} \tag{2}$$

The set of receivers at t_i is represented as:

$$R_{t_i} = \{v \mid v \in P,\ t_i \in T\} \tag{3}$$

Given each directed edge $C_{t_i}(u,v)$, we have the set of communications in t_i:

$$M_{t_i} = \{C_{t_i}(u,v) \mid u \in P,\ v \in P\ \ t_i \in T\} \tag{4}$$

Therefore, the communication graph at t_i is

$$G_{t_i} = (P, M_{t_i}). \tag{5}$$

We then construct a new graph G'_{t_i} to represent the original graph G_{t_i}. We first let $X = P$ and $Y = P$, that is X and Y are the copies of the node set of G_{t_i}. For each edge $C_{t_i}(u,v)$ in G_{t_i}, we construct a corresponding edge $C'_{t_i}(u',v')$ in the new graph, where u' is the copy of u in X, and v' is the copy of v in Y. We denote the union of $C'_{t_i}(u',v')$ as M'_{t_i}. Thus, we obtain a new graph G'_{t_i}:

$$G'_{t_i} = (X \bigcup Y, M'_{t_i}). \tag{6}$$

Lemma. *G'_{t_i} is a bipartite graph that represents the communication information in G_{t_i}.*

Proof: There is a one-to-one mapping between $C_{t_i}(u,v)$ and $C'_{t_i}(u',v')$. Thus, $C'_{t_i}(u',v')$ corresponds to the communication represented by $C_{t_i}(u,v)$ in G_{t_i}. In G'_{t_i}, all the senders are in X, and all the receivers are in Y, that is $S_{t_i} \subset X$ and $R_{t_i} \subset Y$. This means every edge of G'_{t_i} connects a node in X to one in

Y, and X and Y are two disjoint node sets, which satisfies the definition of bipartite graph. ■

Figure 2 (a) shows the bipartite graph constructed from the graph in Figure 1 (a). For a time-varying communication graph, it is possible to construct a bipartite graph for each time step. Alternatively, because X and Y essentially represent the same set of nodes, we can concatenate the bipartite graphs of the consecutive time steps by reusing Y at one time step as X at the next time step. Figure 2 (b) shows an example of the combination of the bipartite graphs at the time steps t_i and t_{i+1} corresponding to Figure 1 (a) and (b). Therefore, we transform a time-varying communication graph into a sequence of bipartite graphs.

E. Visualization Strategies

The new transformation can facilitate our visualization designs to significantly reduce visual clutter and coherently show dynamic communication patterns.

In the original node-link diagram, the directions of edges are rather arbitrary, which may cause a considerable number of crossings between the edges and show the inconsistent directional information. In our new bipartite graph representation, we explicitly duplicate the node set into two disjoint sets, which can force each edge pointing from a set to another set in a roughly consistent direction. Based on this property, we can reinforce the edge directions and make them more visually coherent using the B-Spline technique [24]. The basic idea is to render each edge as a B-spline curve rather than a straight line segment, and the B-spline curves can have a relatively identical orientation.

We can add other visual properties to encode more information. In our design, we first use a bar to represent each node. We then construct the polygons along a B-spline curve, where the width of the polygons is proportional to the message size between the processors. Moreover, we apply a color map and alpha value to different message size. For a processor having multiple simultaneous communications, there are multiple polygons overlapping at the corresponding node. With a naturally alpha blending of polygons, the vicinity of the node will be conveyed with a darker color that is an indicator of a potential communication contention.

Figure 3 shows a butterfly communication pattern that is widely used in many parallel programs [25]. A representative example is the binary-swap image compositing algorithm for parallel rendering [26]. Figure 3 (a) shows the bipartite graph representation of the communication graph on 8 processors, where we can clearly see that each processor exchanges the message exactly with another processor at each step in a binary tree manner. The dynamic communication patterns are displayed concisely over time steps. Figure 3 (b) shows our visualization result. The edges are rendered using the B-spline curve based polygons with the coherent orientations. In this visualization, apart from the communication pattern, we can also observe that the message size exchanged by a processor is reduced by half over every step. This is a key property of binary swap, which is not displayed in Figure 3 (a). Therefore, our visualization can provide an informative and precise description of the communication.

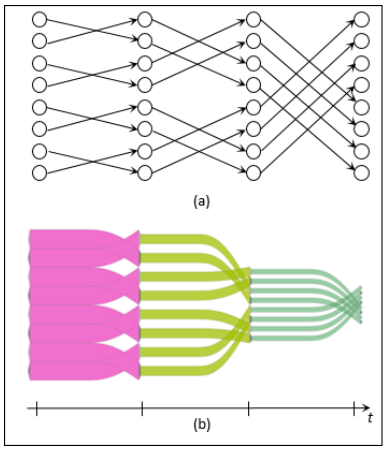

Fig. 3. (a) shows the bipartite graph representation of the communication of the binary swap algorithm on 8 processors. (b) shows the result using our visualization strategies.

F. Temporal Clustering

A real-world communication trace dataset typically contains a large time interval where visual clutter can be generated if we want to show all communication information within a limited display. To address this issue, we propose a new approach to condense communication over time steps.

The key idea is to cluster communication events according to a similarity evaluation. We note that each communication event is represented as an edge e_i which has a pair of sending time and receiving time. We then compute the representative time m_i of the edge e_i using the mean of its sending time and receiving time. Then the similarity between two edges e_i and e_j is defined as

$$d(e_i, e_j) = \mid m_i - m_j \mid . \tag{7}$$

To cluster the edges, we use an agglomerative hierarchical clustering. This is a bottom-up method that begins with each edge in a distinct cluster, and successively merges the two most similar clusters together until only one cluster is left. When we merge two clusters, the representative time of the new cluster is the average of the times of two clusters. During the clustering process, we generate a binary tree to indicate which two clusters are merged at each iteration.

The clustering tree enables a natural definition of levels of detail (LODs) for the communication patterns over time. We can explore communication patterns at various LODs. With the LODs enabled by the clustering tree, we can begin with a few number of cluster at a coarse LOD, which provides us an overview of the communication. If we find some interesting region in the visualization, we can select a finer LOD to generate a zoom-in view to examine the details. For example, Figure 4 shows the communication of binary swap on 16 processors. We can select the number of clusters and conduct a continuous zoom-in of the communication patterns along time.

G. Processor Clustering

The complexity of communication also escalates with the increasing number of processors that participate in message

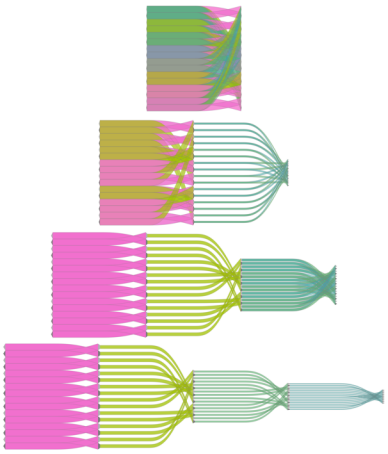

Fig. 4. Temporal clustering for binary swap on 16 processors. We conduct a continuous zoom-into examine the detailed order of the communication events using 1 to 4 time intervals.

exchanges. Visual clutter cannot be avoided with a direct visualization, and we need to reduce the visual complexity to explore the patterns inherent in a large mount of processors.

To solve this problem, we use a graph clustering algorithm [27]. We observe that within a time step the processors can be grouped based on their connections in communication. This problem is similar to the well-known community detection problem in networks [28]. The form of network communities is the gathering of vertices into groups such that there is a higher density of edges within groups than between them. This concept can be applicable to our problem: if we can detect such communities that the processors have a higher amount of messages exchanged within communities than between them, we can place the processors of the same community closer in the visualization, and thus distinguish different communities. In this way, we can minimize the crossings of edges between the communities, and thus reduce visual clutter.

Clustering can be used to detect communities [29]. We use an agglomerative hierarchical clustering algorithm which is proved to work based on modularity [28]. We treat processors as graph nodes and measure modularity gain [27], when the modularity achieves its maximum value, we can build a bottom-up fashion hierarchical tree. Once the clustering is completed, we can use the clustering tree to select the number of clusters. The processors within a cluster form more communication than between clusters, which can provide us a more coherent and appropriate view of communication pattern.

The clustering result helps us address the problem of large processors with a control of LOD. With the cluster tree, we can begin with a few number of clusters to perceive an overview as the color and size of polygons can provide a high-level visual description of communication among the communities. Once we perceive an interesting community, we can select a finer LOD to further partition it into a set of smaller communities and see the detailed communication.

Figure 5 demonstrates our clustering result for binary swap on 16 processors. The results are generated using our

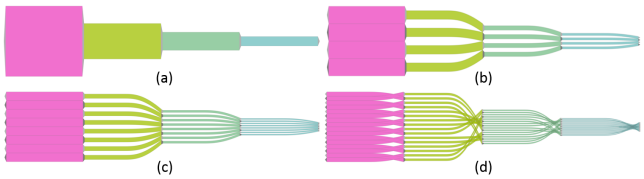

Fig. 5. Processor clustering for binary swap on 16 processors. (a), (b), (d), and (d) show the LOD results with 1, 4, 8, and 16 clusters, respectively, after the bottom-up clustering process. Our method can automatically group the processors into the communication communities, and show the coherent patterns without visual clutter. Our method can achieve the ideal results matching the ground truth of communication pattern without any prior knowledge.

clustering-based community detection method without any prior knowledge of the underneath algorithm. It clearly shows that using hierarchy clustering can not only identify the communication pattern matching the ground truth but also mitigate the visual clutter issue. In addition, even with a smaller number of processors, the communication pattern can be abstracted in a coherent fashion to facilitate our understanding. This mechanism is very useful when the number of processors is large with respect to the available display space.

H. CommGram Curve

With our temporal clustering and processor clustering, our method can generate a visualization with the LOD control according to the time and the number of processors, corresponding to the horizontal and vertical axes in our display space, respectively.

To further enhance our visualization to monitor and diagnose the communication of a very large parallel program, we are inspired by electrocardiograph and construct a curve to generate an even more abstract description of the overall activities of processors. The intuition is that the curve section is rather flat if the communication pattern has not changed over time, while the curve section becomes highly wavy if the communication pattern has changed dramatically. Given this representation, a user can obtain a concise and intuitive description of the variation of communication in a highly complex system, and make timely response if some abnormality has been detected. Then our LOD based visualization (Sections III-F and III-G) can give the user more detailed information for diagnosis. This manner is similar to a doctor keeps watching the patient through the output of an electrocardiogram.

We name our tool as *CommGram*. The curve of CommGram is constructed in a phase wise fashion. We first use the temporal clustering to generate a number of phases, and then apply the processor clustering to detect communities from the communication graph within each phase. It is possible that the processor clustering results can be different between two consecutive phases. Recall that we represent each communication graph as a bipartite graph (Section III-D), we adjust the relative hight of two consecutive bipartite graphs according to the difference between their processor clustering results: the relative hight is zero for two identical clustering results, and a higher value corresponds to two more distinct results. We use the Hungarian algorithm [30] to measure the

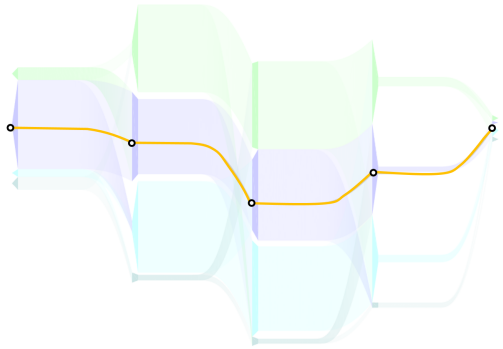

Fig. 6. The CommGram curve (in orange) highlights the variation of communication patterns over time.

difference between two clusters. Figure 6 shows an example of CommGram curve to show the varying communication patterns in a real-world application.

IV. Case Studies

We apply our CommGram tool to analyze the communication data generated from a program that integrates combustion simulation and in-situ visualization [31]. The combustion simulation, S3D [32], has been developed at Sandia National Laboratories to solve the governing equations of direct numerical simulation. During the iteration of simulation, the main communication involves the boundary data exchange among the neighboring processors. In-situ visualization is called at each simulation time step to generate the image of a selected variable. The main communication of visualization is occurred for parallel image compositing, where we use the 2-3 swap algorithm [33]. This algorithm exchanges the messages among the processors in a 2-3 tree manner, and the message number is bounded by $O(N \log N)$ for N processors.

We run the integrated program on Hopper, a Cray XE6 supercomputer at the Lawrence Berkeley National Laboratory. The system contains 53,216 compute cores, 217 Terabytes of memory, and 2 Petabytes of disk. The nodes interconnected by the Cray Gemini Network. We use DUMPI to collect MPI communication trace data from 4320 compute cores. The size of trace data is 216 gigabytes for one iteration of simulation and visualization. The data contains detailed information about communication events, such as senders, receivers, message sizes, communication categories, walltimes and CPU times that a processor starts and finishes a communication event.

We employ our CommGram tool for analysis of such large communication trace data. Figure 7 (a) shows the CommGram curve that provides an overview of communication captured within 3 iterations of simulation and visualization. Repetitive patterns can be clearly perceived from the curve: There are 3 long flat sections and 3 short flat sections, and each section corresponds to a constant communication pattern. A considerable jump between these two types of section means a dramatic change of communication pattern. This curve provide a clear summarization that allows us to effectively monitor the program behavior.

To further examine the details, we use our polygon-based visualization overlaid on the curve, as shown in Figure 7 (b). Initially, we group computing nodes into 18 communities within 27 phases using our temporal and processor clustering. Within each of these phases, the polygons represent the concurrent communication events. Blue color implies a relatively smaller message size while green color indicates a larger one. We can see the pattern within the time intervals t_0 to t_8 is repetitive. This implies that the program is looping for a communication routine. Specifically, the intervals of t_0 to t_7 reveal a similar communication routine which matches the long flat section of the curve in Figure 7 (a). We can clearly see the a big change of communication pattern between t_7 and t_8, which matches the jump in Figure 7 (a) as well. From the visual attributes, it can be perceived that the communities should have more intensive message exchanges in t_8, which can be attributed to three main possibilities:

- A community could have more message exchanges in this phase.
- A community could have larger data to send and receive in this phase.
- A community could have both of the above.

We can testify the possibilities using the zoom-in capability of our tool. Figure 8 shows the detail communications in t_8 that is unfolded into 12 phases with all computing nodes are also grouped into 18 clusters. From this observation, the overall message size is decreasing approximately by half over phases. Note that the data transmitted in each phase are approximately halved in 2-3 swap and it will take $\lfloor log_2 N \rfloor$ stages to complete the image compositing, where $N = 4320$. Therefor, Figure 8 matches our prior knowledge on the communication of 2-3 swap. We continue our investigation by choosing t_{8_0} and increasing the cluster number. Figure 9 shows the result of 4320 clusters. The pattern is an analogy to the first stage in Figure 3 (b). This verifies the correctness of our visualization to 2-3 swap.

We also investigate the long flat section by exploring the detailed patterns in t_0, t_7, and t_9. Figure 10 (a) and (b) show the zoom-in views of t_0 with 24 and 61 clusters, respectively. These patterns are completely different from the result of t_8. Figure 10 (c) and (d) show the communication of a cluster and a node respectively. We can see that each cluster or node sends message to 6 to 9 targets. With prior knowledge, the pattern matches the characteristics of the simulation stage of the source program, where the simulation code uses neighbor exchange among processors. t_7 and t_9 reveal similar patterns as t_0. Hence, in Figure 7 (a), the curve is flat in the first 7 phases, drops down between t_7 and t_8, and rise back before t_9. We justify the result of our visualization with prior knowledge.

V. Conclusions

The performance of parallel programs is heavily affected by communication cost, where each involved processor may exchange and fetch data from the remote memory. Given these behaviors, it is critical for developers to learn the abstract communication pattern from the application. Our CommGram tool, in cooperation with the DUMPI utility of SST, deliveries advanced visual analytics capabilities tailored to the large-scale communication data. Our design fundamentally addresses the scalability problem by carefully characterizing communication

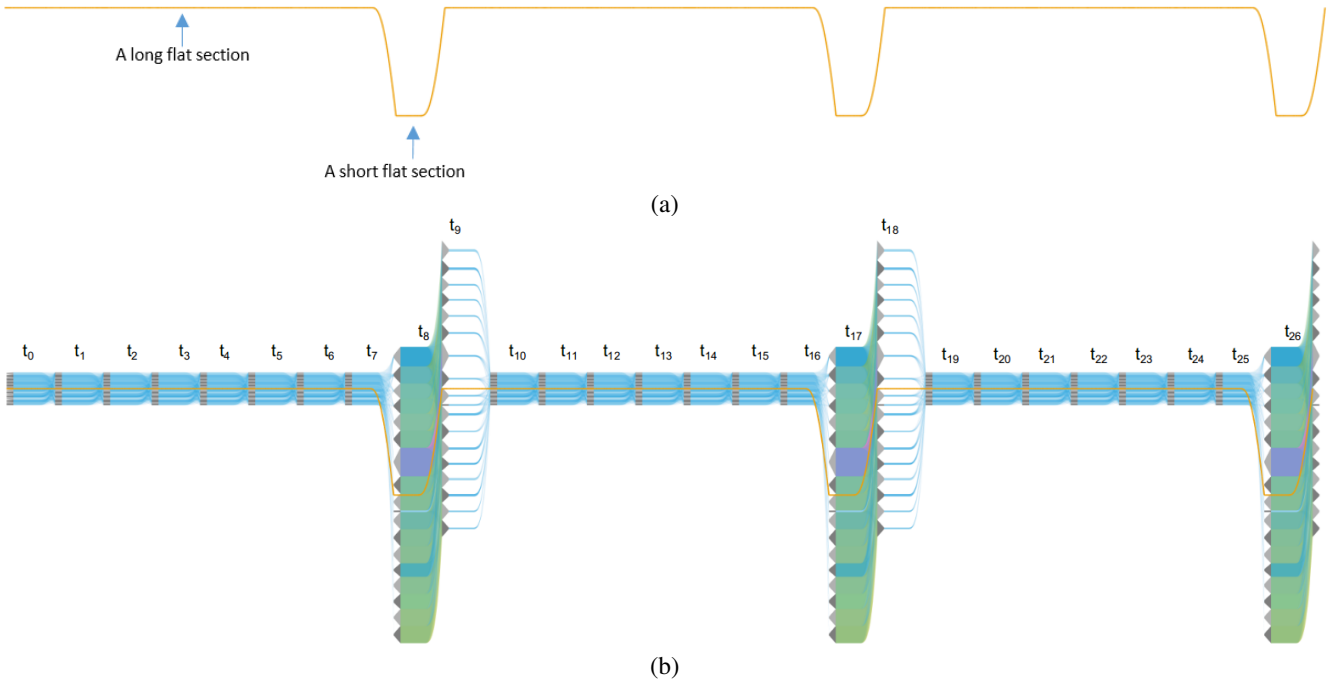

Fig. 7. (a) shows the CommGram curve of the integrated program on 4,320 cores. (b) shows our polygon based visualization.

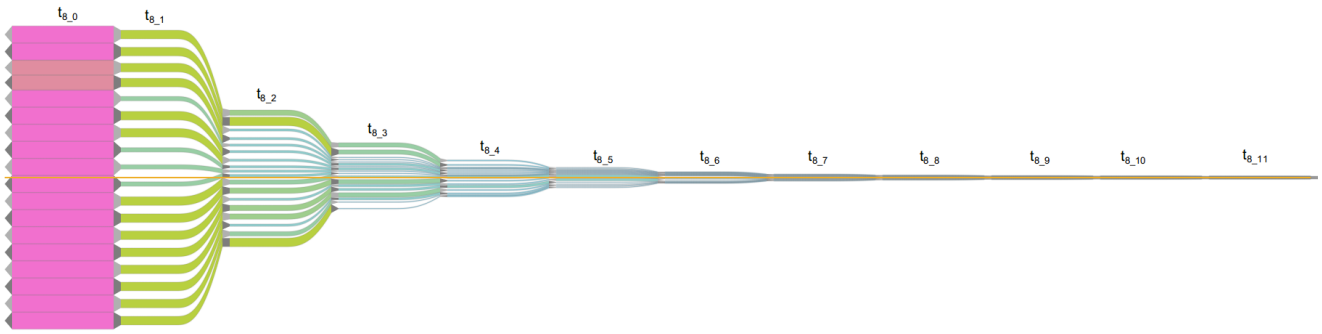

Fig. 8. A zoom-in view of t_8 depicts an analogy to Figure 3 (b) .

Fig. 9. A zoom-in view of t_{8_0} with 4320 clusters, where 1 processor is 1 cluster in this case.

graphs and developing scalable visualization techniques. Our visual representation is easy to understand and simple to use. The effectiveness of our approach has been demonstrated using real-world large applications.

Beyond the inter-node communication, we would like to extend our tool to study the intra-node communication. Data movement pattern within CPU cores or GPU is critical for performance optimization on large heterogeneous systems. We will continue studying techniques to investigate data access at fine-grained thread levels and understand their impact on parallel work processes.

REFERENCES

[1] J. Shalf, S. Dosanjh, and J. Morrison, "Exascale computing technology challenges," in *Proceedings of the 9th international conference on High performance computing for computational science*, ser. VECPAR'10, 2011, pp. 1–25.

[2] W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach, "VAMPIR: Visualization and analysis of MPI resources," *Supercomputer*, vol. 12, pp. 69–80, 1996.

[3] R. Bell, A. Malony, and S. Shende, "ParaProf: A portable, extensible, and scalable tool for parallel performance profile analysis," in *Euro-Par 2003 Parallel Processing*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, vol. 2790, pp. 17–26.

[4] J. S. Vetter and B. R. de Supinski, "Dynamic software testing of MPI applications with Umpire," in *Proceedings of the 2000 ACM/IEEE*

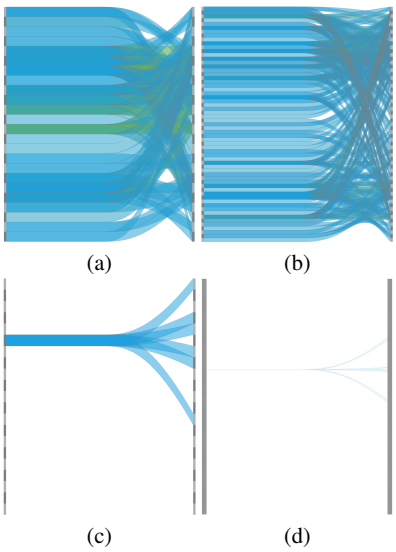

Fig. 10. The zoom-in and selected views of t_0.

Conference on Supercomputing, ser. SC '00. Washington, DC, USA: IEEE Computer Society, 2000.

[5] G. Luecke, H. Chen, J. Coyle, J. Hoekstra, M. Kraeva, and Y. Zou, "MPI-CHECK: a tool for checking fortran 90 MPI programs," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 2, pp. 93–100, 2003.

[6] T. Hilbrich, J. Protze, M. Schulz, B. R. de Supinski, and M. S. Müller, "Mpi runtime error detection with must: Advances in deadlock detection," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 30:1–30:11.

[7] G. Llort, H. Servat, J. González, J. Giménez, and J. Labarta, "On the usefulness of object tracking techniques in performance analysis," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 29:1–29:11.

[8] Y. Sun, G. Zheng, C. Mei, E. J. Bohm, J. C. Phillips, L. V. Kalé, and T. R. Jones, "Optimizing fine-grained communication in a biomolecular simulation application on cray xk6," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 55:1–55:11.

[9] A. Bhatele, T. Gamblin, K. E. Isaacs, B. T. N. Gunney, M. Schulz, P.-T. Bremer, and B. Hamann, "Novel views of performance data to analyze large-scale adaptive applications," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 31:1–31:11.

[10] M. T. Heath and J. E. Finger, "Paragraph: A tool for visualizing performance of parallel programs," Tech. Rep., 1993.

[11] O. Zaki, E. Lusk, W. Gropp, and D. Swider, "Toward scalable performance visualization with jumpshot," *Int. J. High Perform. Comput. Appl.*, vol. 13, no. 3, pp. 277–288, Aug. 1999.

[12] S. S. Shende and A. D. Malony, "The tau parallel performance system," *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 2, pp. 287–311, May 2006.

[13] C. Muelder, F. Gygi, and K.-L. Ma, "Visual analysis of inter-process communication for large-scale parallel computing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1129–1136, Nov. 2009.

[14] C. Muelder, C. Sigovan, K.-L. Ma, J. Cope, S. Lang, K. Iskra, P. Beckman, and R. Ross, "Visual analysis of i/o system behavior for high-end computing," in *Proceedings of the Third International Workshop on Large-scale System and Application Performance*, ser. LSAP '11. New York, NY, USA: ACM, 2011, pp. 19–26.

[15] C. Sigovan, C. Muelder, and K.-L. Ma, "Visualizing large-scale parallel communication traces using a particle animation technique." *Comput. Graph. Forum*, vol. 32, no. 3, pp. 141–150, 2013.

[16] C. Sigovan, C. Muelder, K.-L. Ma, J. Cope, K. Iskra, and R. Ross, "A visual network analysis method for large-scale parallel i/o systems," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, ser. IPDPS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 308–319.

[17] A. Landge, J. Levine, A. Bhatele, K. Isaacs, T. Gamblin, M. Schulz, S. Langer, P.-T. Bremer, and V. Pascucci, "Visualizing network traffic to understand the performance of massively parallel simulations," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 18, no. 12, pp. 2467–2476, Dec 2012.

[18] G. Hendry and A. Rodrigues, "Sst: A simulator for exascale co-design," in *ASCR/ASC Exascale Research Conference*, 2012.

[19] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessors*. Cambridge, MA, USA: MIT Press, 1989.

[20] T. Hoefler and T. Schneider, "Runtime detection and optimization of collective communication patterns," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '12. New York, NY, USA: ACM, 2012, pp. 263–272.

[21] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[22] C. Hurter, O. Ersoy, and A. Telea, "Graph bundling by kernel density estimation," *Comp. Graph. Forum*, vol. 31, no. 3pt1, pp. 865–874, Jun. 2012.

[23] Z. Wang, M. Lu, X. Yuan, J. Zhang, and H. v. d. Wetering, "Visual traffic jam analysis based on trajectory data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2159–2168, Dec. 2013.

[24] X. Yuan, P. Guo, H. Xiao, H. Zhou, and H. Qu, "Scattering points in parallel coordinates," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 6, pp. 1001–1008, Nov 2009.

[25] J. L. Ortega-Arjona, *Architectural Patterns for Parallel Programming: Models for Performance Estimation*. Germany: VDM Verlag, 2009.

[26] K.-L. Ma, J. S. Painter, C. D. Hansen, and M. F. Krogh, "Parallel volume rendering using binary-swap compositing," *IEEE Comput. Graph. Appl.*, vol. 14, no. 4, pp. 59–68, Jul. 1994.

[27] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, p. 066111, Dec 2004.

[28] M. Newman, "Detecting community structure in networks," *The European Physical Journal B - Condensed Matter and Complex Systems*, vol. 38, no. 2, pp. 321–330, 2004.

[29] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Phys. Rev. E*, vol. 76, p. 036106, Sep 2007.

[30] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistic Quarterly*, vol. 2, pp. 83–97, 1955.

[31] H. Yu, C. Wang, R. Grout, J. Chen, and K.-L. Ma, "In situ visualization for large-scale combustion simulations," *Computer Graphics and Applications, IEEE*, vol. 30, no. 3, pp. 45–57, May 2010.

[32] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo, "Terascale direct numerical simulations of turbulent combustion using s3d," *Computational Science & Discovery*, vol. 2, no. 1, p. 015001, 2009.

[33] H. Yu, C. Wang, and K.-L. Ma, "Massively parallel volume rendering using 2-3 swap image compositing," in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, ser. SC '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 48:1–48:11.