Spring 4-2015

Visual Analytics for Large Communication Trace Data

Jieting Wu
University of Nebraska-Lincoln, jtwu1986@gmail.com

VISUAL ANALYTICS FOR LARGE COMMUNICATION TRACE DATA

by

Jieting Wu

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Hongfeng Yu

Lincoln, Nebraska

May, 2015

VISUAL ANALYTICS FOR LARGE COMMUNICATION TRACE DATA

Jieting Wu, M.S.

University of Nebraska, 2015

Adviser: Hongfeng Yu

Executions of modern parallel programs often yield complex communications among compute nodes of large-scale clusters of workstations or supercomputers. Analyzing communication patterns is becoming increasingly critical to performance optimization. As the scale and complexity of parallel applications drastically increases, visualization has become a feasible means to conduct analysis of massive communication patterns. However, most visualization tools fall short in showing comprehensive dynamic communication graph and addressing the scalability issue. Our solution for analyzing dynamic communication patterns is based on an analytics framework coupled with a new visualization technique, named *CommGram* [29], that provides a flexible solution to the scalability issue. We can explore large communication data at different levels of detail, and detect potential communication bottlenecks of massive parallel programs. The conclusion of our studies is based on large-scale scientific applications that include end-to-end simulation pipelines and AMR-based simulations.

ACKNOWLEDGMENTS

Professor Hongfeng, my advisor, is on top of my appreciation and acknowledgement list. Through his endless support, guidance, feedback and critique throughout my graduate work, I have achieved many of my goals and also developed a broad set of skills ranging from scientific thinking, to programming and data analysis.

I would appreciate the constructive comments and valuable guidance from our colleauges: Jianping Zeng, Lina Yu, Shruti Daggumati, Yanfu Zhou. In addition, I would also thank Jianping Zeng for helping me with implementations in this research.

I would also like to thank Professor David Swanson and Professor Ying Lu for their time, guidance, patience and insights on my graduate study as committee members.

I would like to thank the one who I love. This love goes beyond words, distance, and time.

At the last, I would also like to thank all my friends and my family for their encouragement and support whenever I faced challenges.

This research has been sponsored by the Department of Energy through the ExaCT Center for Exascale Simulation of Combustion in Turbulence.

Contents

Contents **iv**

List of Figures **vi**

1 Introduction **1**

2 Related work **4**

3 Visual Analytics Framework **7**

3.1 DUMPI: Communication Trace Data Collection 7

3.2 CommGram: Communication Trace Data Visualization 8

3.2.1 Characterization of Communication Graph 9

3.2.2 Equivalent Transformation of Communication Graph 10

3.2.3 Visualization of Bipartite Graphs 12

3.2.4 Temporal Clustering 13

3.2.5 Processor Clustering 15

3.2.6 CommGram Curve Generation 17

3.3 Visual Exploration 18

4 Case Studies **21**

4.1 An End-to-end Simulation Pipeline 21

4.2 AMR-based Simulations 26

5 Conclusion **33**

Bibliography **35**

List of Figures

1.1 Einthoven's electrocardiogram 2

3.1 The overview of the framework 8
3.2 Equivalent Transformation of Communication Graph 9
3.3 Visualization result of binary swap algorithm 13
3.4 Visualization result of time clustering 15
3.5 Visualization result of processors clustering 17
3.6 CommGram Curve 19
3.7 GUI 20

4.1 In-situ simulation and visualization 22
4.2 CommGram result of the end-to-end simulation pipeline 23
4.3 Zoom-in view of the CommGram 25
4.4 Zoom-in view of the CommGram 25
4.5 Visualization of 2-3 swap with asynchronized communication 26
4.6 Zoom-in view of the asynchronized communication 26
4.7 CommGram result for AMR simulations 27
4.8 CommGram result for advection-diffusion simulation 28
4.9 The polygon-base visualization for advection-diffusion simulation 28
4.10 Detail view of the advection-diffusion simulation 29

4.11 Detail view of the advection-diffusion simulation 30

4.12 The CommGram result for the polytropic simulation 31

4.13 The polygon-base visualization for the polytropic gas simulation 31

4.14 Detail view of the polytropic gas simulation 32

Chapter 1

Introduction

As the scale and complexity of modern high performance computing (HPC) systems increases, performance analysis has become ever more crucial to ensure the efficiency and scalability of parallel programs. The nature of parallel programs affirms that communication among compute nodes is the key in performance analysis. While there exist several HPC-based tools to collect communication trace data, interpreting such large data is very challenging mainly because of three reasons:

First, the sheer size of trace data simply becomes a challenge. A supercomputer can consist of hundreds of thousands of compute nodes, each of which is made up of several cores, introducing exponential growth in explicit parallelism. Thus, tracking comprehensive communication at different details from a large application (such as a tera- or peta-scale scientific simulation) can generate trace data with a size possibly comparable to the data generated by the application itself.

Second, mining dynamic distributed communication patterns and their variations is challenging. For most parallel data-dependent problems, the communication is heavily affected by the dynamic distribution of data and thus is less predictable using prior knowledge. Even for the programs that have static communication patterns, the real trace data may exhibit completely different patterns due to dynamic run-time factors, such as node failure, job scheduling, and so on, which makes it difficult to understand the program behaviors from the trace data. Even though some prior work uses logical order to proceed the event ordering, they only conduct their experiment on a modest data set. As the number of computing nodes increases, the effectiveness of a visual result decreases. We attribute this phenomenon to the scalability issue.

Third, fundamentally, communication events can be represented as communication graphs. However, large time-varying graph analysis remains an open and challenging problem, and there is still a lack of scalable solutions.

Visualization can help interpret trace data and identify communication patterns. Several sophisticated analysis tools, such as Vampir [18] and TAU [23], are equipped with visualization algorithms to exhibit the process of each compute node and the communication among nodes. However, these tools cannot avoid visual clutter resulting from edge crossing as the number of nodes increases, and fail to comprehensively present time-based dynamic communication graphs. Coping with visual clutter, temporal coherence, and scalability issue is becoming a severe barrier for analysts and developers.

Figure 1.1: Einthoven's electrocardiogram recordings [6].

This research presents the design of a visual analytics framework for large communication data generated from massive parallel applications. Our work is inspired by the electrocardiogram that shows the heart's electrical activity in various curves. Figure 1.1 shows the first electrocardiogram recorded by Einthoven, which can reflect possible diseases resulting in changes in heart muscle, and has been widely used as a diagnostic test in today's clinical medicine. This method facilities immediate practical monitoring and analyzing of heart diseases in a simple and concise way [6].

Our framework incorporates a new visualization technique *CommGram* [29] that, like electrocardiogram, generates curves to visually convey communication activity of a massive parallel program. In addition, our framework uses a MPI trace library, named *DUMPI*, to intercept communication activity from a massive parallel program. Our framework has the following major advantages. First, DUMPI can intercept detailed communication activity from applications. Second, CommGram can concisely and coherently display dynamic communication activity, mitigating visual clutter and scalability issues. Third, CommGram accentuates visual foci and can facilitate the detection of potential communication abnormalities. Finally, CommGram is generated from a hierarchical abstract, and can capture communication activity at various levels of detail (LODs). We have used real-world large-scale applications to verify the effectiveness and comprehensiveness of our framework. The applications include end-to-end simulation pipelines and AMR-based simulations. We show that our framework provides informative depictions of communication in a coarse-to-fine manner, and is helpful for us to monitor and diagnose complex communication activity in parallel processing, and gain an insight of communication characterization.

Chapter 2

Related work

There is a substantial amount of techniques and tools that aid parallel program analysis. Some early studies ParaProf [3], Umpire [27] and MPI-Check [15] worked on MPI programming. To detect processing deadlock, Hilbrich et al. [9] developed a graph-based approach visualization tool named MUST. Llort at al. [14] clustered and summarized the performance information to convey evolution of a parallel application along multiple execution scenarios. The framework uses object tracking techniques for performance analysis.

Besides these general techniques, researchers and developers also conducted studies on analysis techniques in order to make performance improvements for large applications. Sun et al. [26] leverage a Projections method to examine and optimize fine-grained communication in a biomolecular simulation application and make a remarkable improvement on a Cray XK6 system. A binary tree structure is utilized by Bhatele et al. [4] to visualize communication topology which establishes monitoring of workload distribution and communication. They showed the performance improvement for a large AMR application on an IBM Blue Gene/P system.

Analysts and researchers often utilize visualization for performance analysis. The typical visualization includes Gantt chart, node-link diagram, adjacency matrix, Kiviat diagram. Some of these designs are sometimes combined to provide an insight from multiple perspectives. ParaGraph [7], an early work of performance visualization, has incorporated multiple types of typical visualizations. Particularly, the tool represents a parallel system by a graph, where the nodes represent processors, and the arcs represent communication between processors. The dynamic status of communication is showed by an animation of the graph presentation.

Jumpshot [31] and Vampir [18] use a sophisticated HPC library to collect the MPI communication events and display an overview of system activities. MPI communication events are presented as a Gantt chart. Different colors are equipped to show different types of MPI calls, while arrows implies the communication direction between two corresponding processes. An analog of these two tools is TAU [23]. It also uses a similar Gantt chart. In addition, TAU employs an adjacency matrix-based visualization to depict the complete communication between every pair of nodes.

These conventional visualizations can intuitively depict a communication graph. But they all have inherent disadvantages. Node-link diagrams and matrix based visualizations are not flexible to show the dynamic change of the communication graph over time. Considering temporality is a significant attribute of communication graph in parallel processing analysis, node-link diagrams and matrix based visualizations are less competitive in this aspect. On the other hand, Gantt chart based techniques suffer from the scalability problem. The drastically increasing size of computing nodes in parallel system incurred severe visual clutter. Such visual clutter can interfere with the resulting output that may affect the human perception of the pattern appearance. To address this issue, Muelder et al. [17] utilized high-precision alpha blending and opacity techniques to optimize the visual representations of communication event. The resulting output forms a clear pattern with low visual clutter, but some detailed information can be absent. Sigovan et al. [24] made great effort to extend the overplotting method for large communication graph visualization. They invented a particle animation technique that can display every event in trace data while maintaining an overview of system activity. Also, Sigovan et al. [25] introduced a visual analysis framework for the trace data from large parallel I/O system. In the perspective of infrastructure, Landge et al. [13] applied a cube shaped torus to the network traffic of a supercomputer. It provides a combination of 2D and 3D view in support with large-scale underlying network simulations.

Chapter 3

Visual Analytics Framework

We present a visual analytics framework that provides a new perspective to HPC communication analysis. Figure 3.1 sketches the overall software architecture, which consists of three main components: DUMPI, CommGram, and Exploration. We use DUMPI to capture detailed communication data from parallel programs at runtime, and develop CommGram to generate a concise and informative visualization of communication data based on a novel graph transformation. The new visualization enables an interactive exploration of complex communication patterns at various LODs. In the following, we describe these components in detail.

3.1 DUMPI: Communication Trace Data Collection

DUMPI is a MPI trace library of the Structural Simulation Toolkit (SST) [8], which facilitates the collection and accessing of communication trace data of MPI applications. DUMPI uses the PMPI interface to intercept MPI calls, which is similar to the method for Open Trace Format (OTF) [11]. Moreover, it also monitors the start

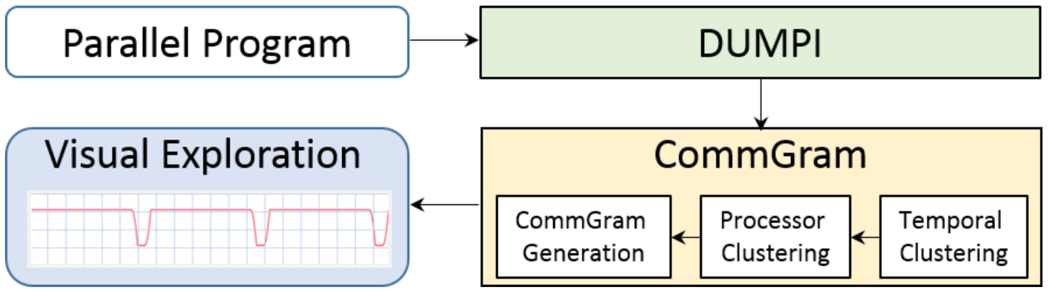


Figure 3.1: The architecture of our visual analytics framework.

and end value of MPI functions and MPI requests at runtime. The output of DUMPI involves exhaustive information of all MPI function calls, such as sender, receiver, message size, time stamps, and so on. We can filter the DUMPI output to capture the specific peer-to-peer communication functions (MPI-Send, MPI-Recv, MPI-Isend, MPI-Irecv, etc.) and the particular collective functions (MPI-Gather, MPI-Reduce, etc.). With DUMPI, the communication activity from a MPI application can be faithfully captured.

3.2 CommGram: Communication Trace Data Visualization

Communication events are typically represented as graphs. Traditional visualization techniques, such as node-like diagrams, adjacency matrixes, and Gantt charts, routinely suffer from temporal incoherence and/or visual clutter in coping with dynamic and scalability issue. CommGram advocates a design that is formally tailored to the characteristics of communication graphs.

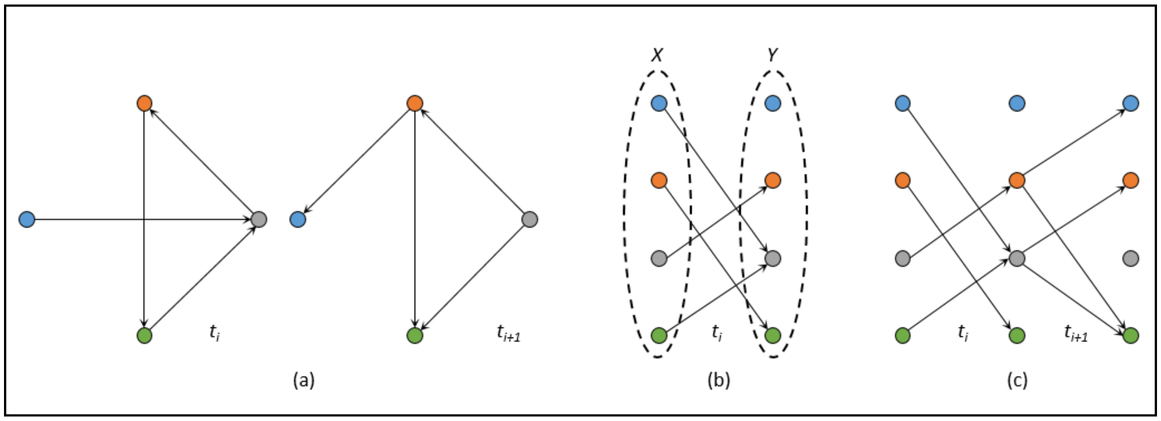


Figure 3.2: (a) shows the traditional node-link diagrams for two consecutive time steps. (b) shows the bipartite graph equivalent to the communication graph of t_i in (a). (c) shows that we can concatenate the bipartite graphs over the time steps to represent the time-varying communication graph.

3.2.1 Characterization of Communication Graph

In a communication graph, each vertex denotes a processor and each directed edge represents a message sent from one processor (sender) to another (receiver). Compared to a general graph, a communication graph possesses a set of exclusive characteristics. First, a communication graph is typically modeled as directed acyclic graph (DAG) [22], or more particularly, communication directed graph (cDAG) [10]. Second, a communication graph is not directly related to the network flow problem [2], in that, the amount of messages received by a processor has no necessary relationship to the amount messages sent by this processor.

However, these characteristics have been overlooked in existing visualization studies. For instance, with the popular node-link diagram, a visualization may show an artifact of circles in the communication graph and cause potential misunderstanding, as shown in Figure 3.2 (a). Moreover, most prevalent graph visualization techniques are developed based on network flows (e.g. [28]), which are not suitable for communication graphs.

3.2.2 Equivalent Transformation of Communication Graph

The design of CommGram is based on an equivalent transformation where a sequence of bipartite graphs are used to represent a time-varying communication graph. Assume we have a time-varying communication graph G over a total of T time steps with n processors. The processors can be described as a set P of nodes, and the edges are regarded as communication among them. 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:

$$C_{t_i}(u, v) = \{\overrightarrow{uv} \ \mid \ u \in P, \ v \in P, \ t_i \in T\} \tag{3.1}$$

The set of senders at t_i is represented as:

$$S_{t_i} = \{u \ \mid \ u \in P, \ t_i \in T\} \tag{3.2}$$

The set of receivers at t_i is represented as:

$$R_{t_i} = \{v \ \mid \ v \in P, \ t_i \in T\} \tag{3.3}$$

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

$$M_{t_i} = \{C_{t_i}(u, v) \ \mid \ u \in P, \ v \in P \ \ t_i \in T\} \tag{3.4}$$

Therefore, the communication graph at t_i is

$$G_{t_i} = (P, M_{t_i}). \tag{3.5}$$

We construct a new graph G'_{t_i} to represent the original graph G_{t_i}. We let X and Y be the copies of the node set of G_{t_i}, and X is regarded as sender while Y is receiver. 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}:

$$G'_{t_i} = (X \bigcup Y, M'_{t_i}). \tag{3.6}$$

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

Proof. There is a one-to-one mapping between $C_{t_i}(u, v)$ and $C'_{t_i}(u', v')$ $C'_{t_i}(u', v')$ 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 sets, which satisfies the definition of bipartite graph. $\square$

Figure 3.2 (b) shows the bipartite graph constructed from the graph of one time step in (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 3.2 (c) shows an example of the combination of the bipartite graphs at the time steps t_i and t_{i+1} corresponding to Figure 3.2 (a) and (b). Therefore, we equivalently transform a time-varying communication graph into a sequence of bipartite graphs.

3.2.3 Visualization of Bipartite Graphs

The conceptual bipartite graph structure provides us a new way to possibly reduce visual clutter and appropriately show dynamic communication patterns. We introduce several visual properties and metaphors to visualize bipartite graphs. In our design, every graph node is represented as a bar. To mitigate visual clutter resulted from edge crossing, we render each edge as a B-spline curve such that the directions of edges can be artificially bent to become more visually coherent. A polygon is constructed along a B-spline curve, where the width of polygon is proportional to the message size between the processors. Moreover, we also apply a color and opacity map to different message sizes. A darker color and a higher alpha value of a polygon indicate a larger amount of message exchanged and vice versa. Alpha value also helps make a coherent visual result that the polygons overlapping at the vicinity of the processor can still be distinguished. It implies this processor is sendering or receiving muliple messages simultaneously. Naturally, the overlapping area also implies a potential communication contention of the incident processor.

Figure 3.3 shows a butterfly communication pattern widely used in many parallel programs [20]. A representative example is the binary-swap image compositing algorithm for parallel rendering [16]. Figure 3.3 (a) shows the bipartite graph representation of the communication on 4 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 aligned concisely over time steps. Figure 3.3 (b) shows our visualization result. The edges are rendered using the B-spline curve based polygons with the coherent orientations. In this visualization, 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

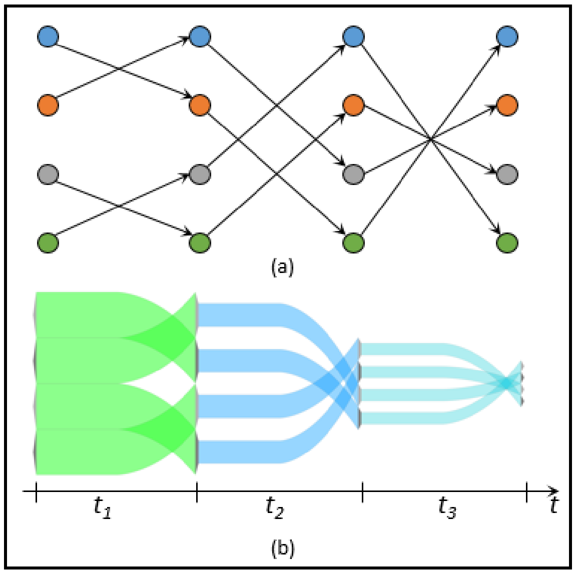


Figure 3.3: (a) shows the bipartite graph representation of the communication of the binary swap algorithm on 4 processors. (b) shows our visualization result.

displayed in Figure 3.3 (a). Therefore, our visualization can provide an informative and precise description of the communication.

3.2.4 Temporal Clustering

In a real-world communication trace data, the communication events or steps may not be clearly distinguished and perfectly aligned over time, compared to the ideal case shown in Figure 3.3. To address this issue, we 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 the receiving time and sending time. The similarity between two edges e_i and e_j is thus defined as the difference of the representative time between two edges:

$$d(e_i, e_j) = \mid m_i - m_j \mid . \tag{3.7}$$

In practice, we do not need to compute the difference between all pairs of edges. This is because we can first sort the edges according to their representative times, and then for an edge, its most similar edge only can be one of its immediate neighbors.

We use an agglomerative hierarchical clustering method to cluster the edges in a binary tree manner. Initially, each edge is a distinct cluster. We then iteratively merge two most similar clusters, where the representative time of the new cluster is the average time of the two merged clusters. An attribute, *step*, is associated with the cluster and records the step of iteration at which this cluster is generated. The iteration stops when only one cluster is left. By default, $step = 0$ for all leaves and $step = Q - 1$ for the root, where Q is the total number of edges.

After we generate the clustering tree, we can select the number of clusters according to the step order in the binary tree. If we want q clusters, we can partition the binary clustering tree into q sub-trees. The root r of a sub-tree satisfying:

$$step_r \leq q \;\; and \;\; step_{p(r)} > q, \tag{3.8}$$

where $step_r$ is the step order of r, and $p(r)$ is the parent of r. By choosing different number of clusters, we can explore the communication graph at different LODs. We can begin with a small number of cluster at a relatively coarser LOD, which depicts a higher-level overview of communication. Then we can select a finer LOD to generate a more detailed view for analysis if some intriguing pattern shows up. Figure 3.4 shows an example of binary swap on 8 processors, where we conduct a continuous zoom-in along time.

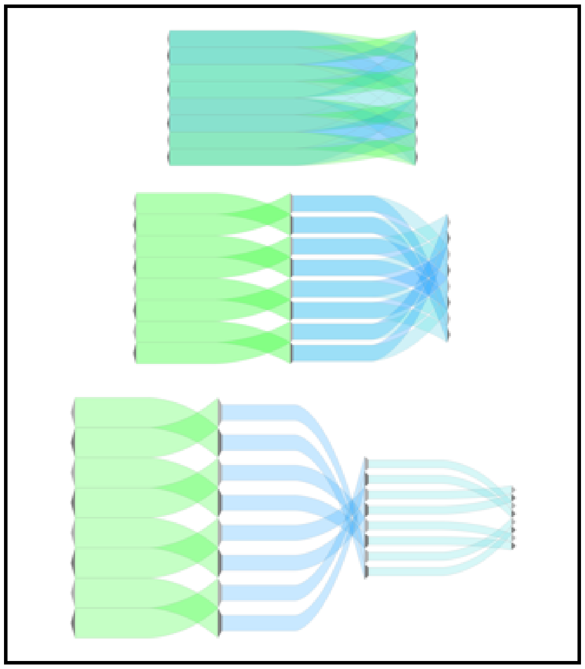


Figure 3.4: By leveraging the clustering tree, we can incrementally increase the number of clusters along time. The top image shows all communication of binary swap on 8 processors in one time step. The middle and bottom images show two and three time steps, respectively. The bottom image clearly depicts the communication pattern of binary swap.

3.2.5 Processor Clustering

Solving the temporal alignment problem is still not sufficient to generate an appropriate view of communication pattern as the ideal result in Figure 3.3. There remains two key problems.

First, the ideal result in Figure 3.3 is generated based on an assumption that we already know a butterfly-style communication has been used in programming, and thus arrange the processors in pairs at each stage. However, in a real-world case, we may have no prior knowledge about how the processors exchange messages.

Second, the complexity of communication also escalates with the increasing number of processors that participate in message exchanges. Visual clutter cannot be avoided with a direct visualization of communication. It is paramount that visualization should reduce the complexity of the visual result in order to increase the effectiveness of exploring the patterns inherent in a large amount of processors.

Our solution to these two problems is also based on clustering. 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 [19]. 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.

We use an agglomerative hierarchical clustering algorithm to detect communities [21], which is based on modularity [19]. Processors are treated as graph nodes while communications are considered as edges. At first, one processor is in one community. Then we measure the modularity gain [5] to iteratively cluster distinct communities until only one community is left. Similar to temporal clustering, a clustering tree is generated and used to select the desired clusters. The processors in one community have more communication than between communities. Using the hierarchical structure enables the zoom-in functionality into a community. We can have a flexible and adjustable overview to the communication result, and dive into the communities to explore the detailed message exchanges if interesting patterns are perceived.

Figure 3.5 demonstrates our clustering result for binary swap on 8 processors. The results are generated using our clustering-based community detection method without any prior knowledge of the underneath algorithm. It clearly shows that this

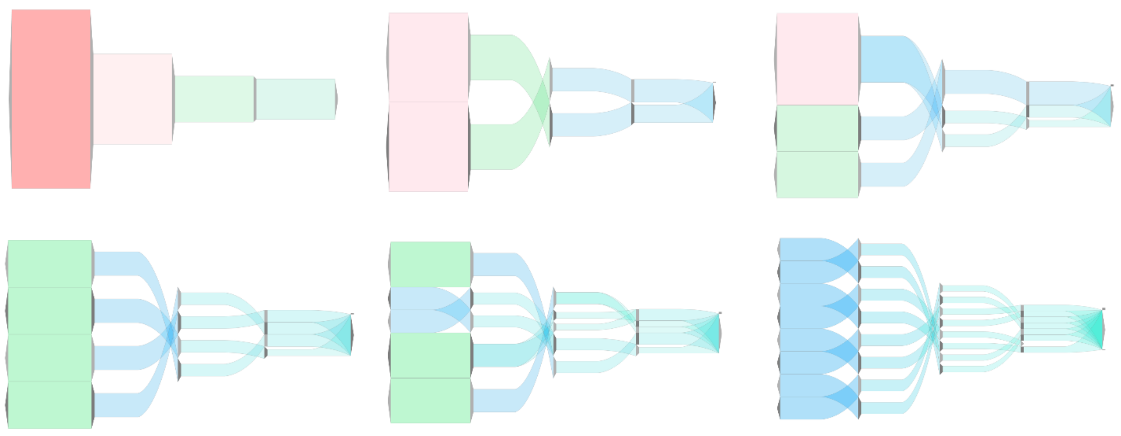


Figure 3.5: Processor clustering for binary swap on 8 processors. After the bottom-up clustering of processors, we can incrementally increase the number of communication communities. The image shows the LOD results with 1, 2, 3, 4, 5, and 8 clusters, respectively. Our method can automatically group the 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.

method can perfectly identify the communication pattern matching the ground truth. 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 also very useful when the number of processors is large with respect to the available display space.

3.2.6 CommGram Curve Generation

We use temporal clustering and processor clustering to generate a visualization with LOD controls for time and processors respectively. To further detect communication abnormality, monitoring the communication over time is necessary. In this work, we focus on the variation of communication patterns. Inspired by the electrocardiogram, we construct a curve that provides the description of pattern variation. The intuition is that the curve section is relatively flat if the pattern has been stable over time while the highly wavy section of the curve implies the communication pattern has changed drastically. Performance analysis can benefit from this representation such that analysts can have an intuitive and concise description of the changes of communication over time in a highly complex and dynamic parallel system. Generally, the highly wavy sections are more likely to be the abnormality that could visually intrigue the analysts. Then the LOD based visualization control can provide the analysts more details for diagnosis, which is an analog to electrocardiogram.

The curve of CommGram is constructed in a phase-wise fashion. Recall that, with temporal clustering and processor clustering, we obtain a time-varying communication graph consisting of a set of consecutive bipartite graphs. To measure the difference between two bipartite graphs in distinct time steps, we use the Hungarian algorithm [12]. We treat the neighboring time step as a perfect matching bipartite graph where the vertex denotes community, and the edge with a weight represents the difference of the two incident communities. Then we apply the Hungarian algorithm to find the perfect matching of minimum difference. The result is the sum of minimum difference in this bipartite graph. Visually, we adjust the relative height of two consecutive bipartite graphs according to the maximum difference. It can be interpreted that the two identical clustering results have the same height in the representation and vice versa. In general, a time-varying graph has a certain fluctuation if the communication pattern changes, as shown in Figure 3.6.

3.3 Visual Exploration

We develop a graphical user interface (GUI) to support a visual exploration of communication data. A set of interaction methods can be utilized to show the different insights of communication patterns. We first provide a user controller to adjust the

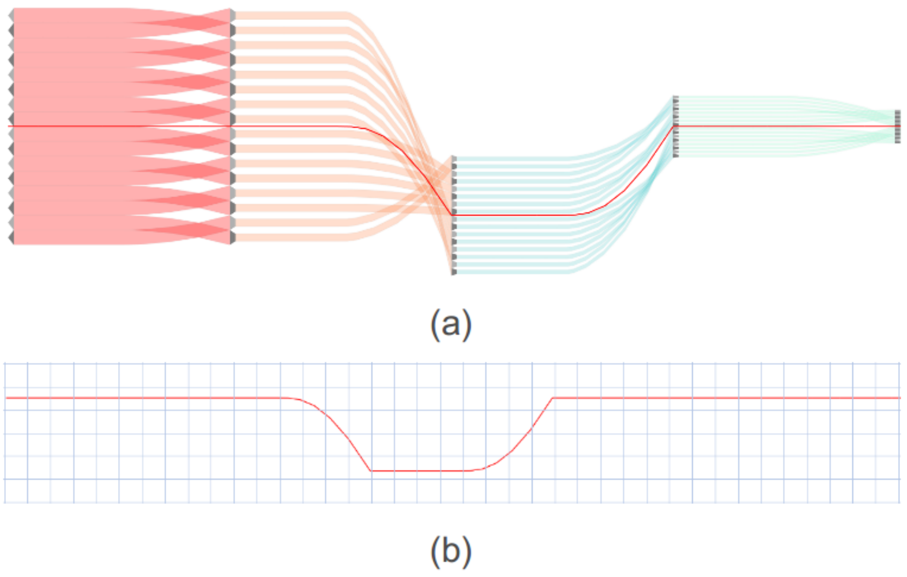


Figure 3.6: (a) shows the relationship between the CommGram curve (in red) and the bipartite graphs. (b) only shows the CommGram curve that gives an overview of variation of communication patterns over time in a simple and concise way.

polygon with respect to the message size. When a user moves a mouse over or touches a polygon, the detailed information of communication (such as MPI operations, senders/recievers, message sizes, etc.) will be popped up. We also allow users to control the relative hight between two consecutive bipartite graphs, and adjust the wavy degree of the CommGram curve, which can help users detect more interesting patterns. More importantly, users can select the number of processor clusters to examine different details across all time steps. It provides a natural mechanism for investigating different aspects of the general graph. Besides, a sub-window is implemented to show one specific cluster communication result with the zoom in and out functionality along the vertical directions, which enables us to implement touch-based interaction, as shown in Figure 3.7.

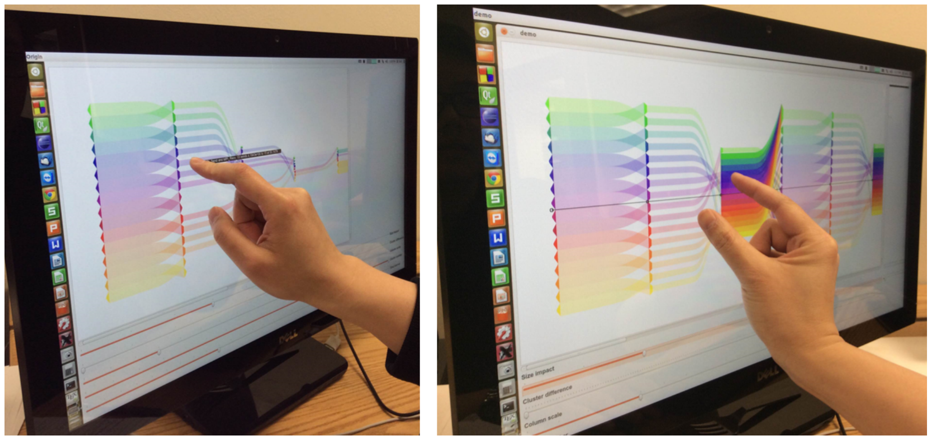


Figure 3.7: A GUI overview shows an interactive exploration of communication data on a touch screen using our CommGram tool.

Chapter 4

Case Studies

We first apply our framework to analyze the communication data generated from large-scale applications including an end-to-end simulation pipeline and AMR simulations. The applications are performed 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 are interconnected by the Cray Gemini Network. We use DUMPI to collect detailed MPI communication trace data, and employ CommGram to visualize data for analysis.

4.1 An End-to-end Simulation Pipeline

We first apply our framework on an end-to-end simulation pipeline. The pipeline tightly couples combustion simulation and in-situ visualization that are executed on the same set of processors. Each iteration or time step of simulation requires an exchange of boundary data among the neighboring processors. In-situ visualization is called after each time step of simulation to render the selected variables. The main communication of visualization is required for the 2-3 swap image compositing

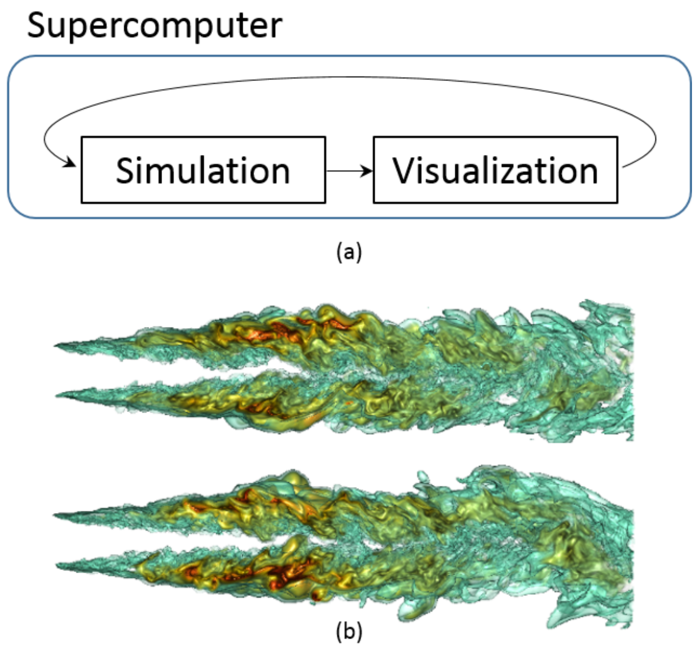
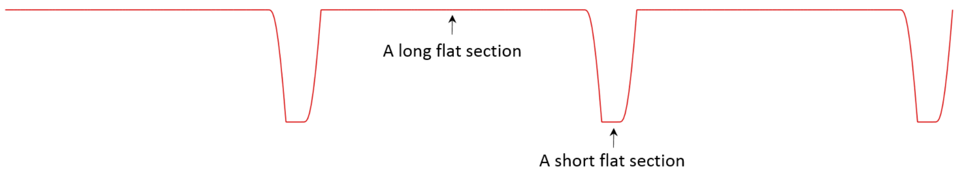


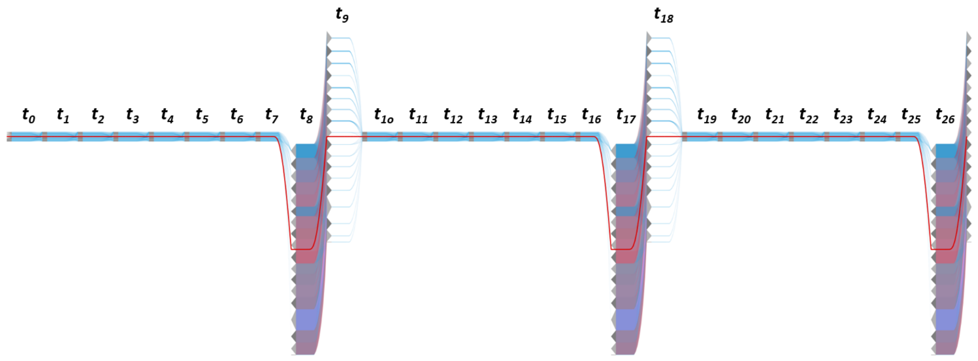
Figure 4.1: (a) shows the framework of an end-to-end pipeline that integrates simulation and visualization. (b) shows the images of two time steps generated by visualization during a simulation run.

algorithm [30] in support of massively parallel rendering. This image compositing 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. Binary swap is a special case of 2-3 swap when N is power-of-two. Figure 4.1 (a) shows the framework of the end-to-end simulation pipeline, and (b) shows the images of two time steps generated by in-situ visualization. This simulation pipeline allows scientists to monitor the simulation at runtime with a high frequency and possibly capture highly intermittent transient phenomena.

We run the integrated simulation and visualization on 4320 processors, and use DUMPI to collect MPI communication trace data. The size of trace data is about 216 gigabytes for one iteration of simulation and visualization. The data contains detailed information about communication events, such as senders, receivers, message



(a)


t_9
t_{18}
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8
t_{10} t_{11} t_{12} t_{13} t_{14} t_{15} t_{16} t_{17}
t_{19} t_{20} t_{21} t_{22} t_{23} t_{24} t_{25} t_{26}

(b)

Figure 4.2: (a) shows the CommGram curve of the end-to-end simulation pipeline on 4,320 processors. We can clearly see two completely different periodic communication patterns, corresponding to simulation and visualization, respectively. (b) shows our polygon based visualization of bipartite graphs.

sizes, communication categories, walltimes and CPU times that a processor starts and finishes a communication event.

We use our visual analytics framework to process this large communication data. Figure 4.2 (a) shows the CommGram curve that provides an overview of communication captured within 3 iterations of simulation and visualization. We can clearly see repetitive patterns from the curve: there are 3 long flat sections and 3 short flat sections, where each section corresponds to a constant communication pattern. A jump between these two types of sections implies a dramatic change of communication pattern. We can intuitively know that these two sections correspond to the communication of simulation and visualization, respectively. This curve clearly summarizes the program communication activity and enables an effective monitoring.

Our framework also provides more detailed visualization of communication by overlapping polygon-based visualization with the CommGram curve, as shown in Figure 4.2 (b). As described in Section 3.2.3, each polygon represents a communication event, where light blue indicates a relatively smaller message size, and dark red implies a larger message size or more message exchanges. We use the temporal and processor clustering methods to group the compute node into 18 communities within 27 phases. We can clearly see the patterns between t_0 to t_8 is repeated at the intervals of t_9 to t_{17} and t_{18} to t_{26}. Within t_0 to t_8, the patterns between t_0 to t_7 are nearly identical, implying a similar communication routine. However, there is a complete difference between t_7 and t_8, where t_8 shows an intensive message exchanged among the processors, which matches the big jump between the long flat section and the short section in Figure 4.2 (a).

We can further investigate the communication in each phase. We first zoom into t_8 and unfold it into 12 phases with 18 communities, as shown in Figure 4.3. We can clearly see that the message size is decreasing approximately by half over phases. Recall that 2-3 swap takes $\lfloor \log_2 N \rfloor$ stages to complete image compositing, where $N = 4320$ in this case, and the message size is approximately halved in each stage of image compositing, which is reflected in Figure 4.3. To verify the communication pattern of 2-3 swap, we further zoom into t_{8_0} and t_{8_1} by displaying all 4320 processors, as shown in Figure 4.4. They show an analogy to the binary swap pattern as shown in Figure 3.3 (b), thus verifying the correctness of our visualization of 2-3 swap.

In real-world parallel programming, the communication of image compositing can be either synchronized or asynchronized among the processors: with synchronized communication, the synchronization function of MPI is called after each stage of image compositing, while this function is deprecated in asynchronized communication. Figure 4.3 shows the result of synchronized communication. We also collect and visualize the data of asynchronized communication, as show in Figure 4.5. We can

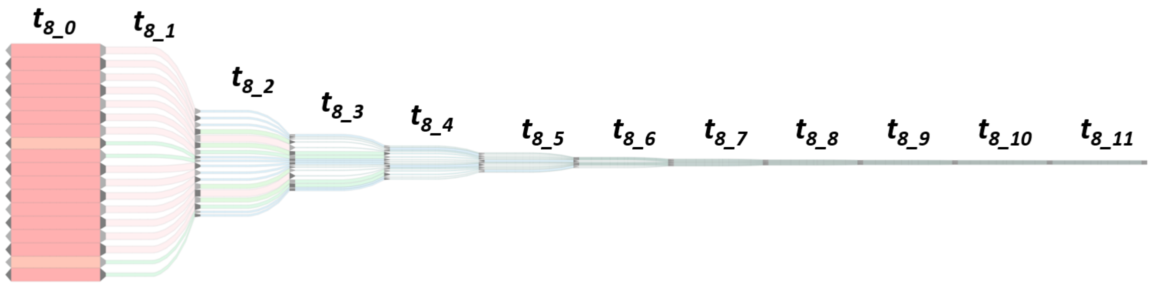


Figure 4.3: A zoom-in view of t_8 in Figure 4.2 (b), unfolding it into 12 phases with 18 communities.

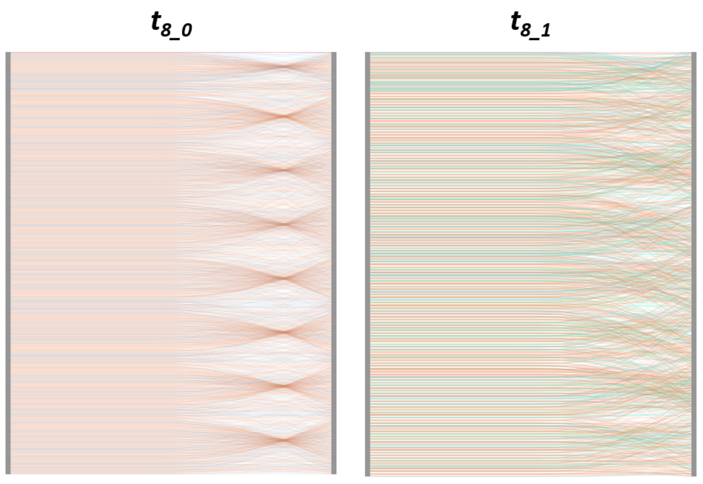


Figure 4.4: A zoom-in view of t_{8_0} and t_{8_1} in Figure 4.3 by displaying all 4320 processors.

see that the message size is still reducing over the stages, and the trend complies with 2-3 swap. However, compared to Figure 4.3, the message size of each community is not even in each stage of Figure 4.5. In particular, four intensive communication events have be spotted in t_{8_0}, while communication is relatively sparse in t_{8_1}, t_{8_2}, and t_{8_3}. This implies that some message exchanges of a later stage may occur in the first stage due to a lack of synchronization. Figure 4.6 shows a zoom-in view of t_{8_0} in Figure 4.5, and display the roughly same communication pattern as Figure 4.4. However, compared to t_{8_0} in Figure 4.4, Figure 4.6 shows more communication events in green, indicating that more communication events with smaller message sizes have

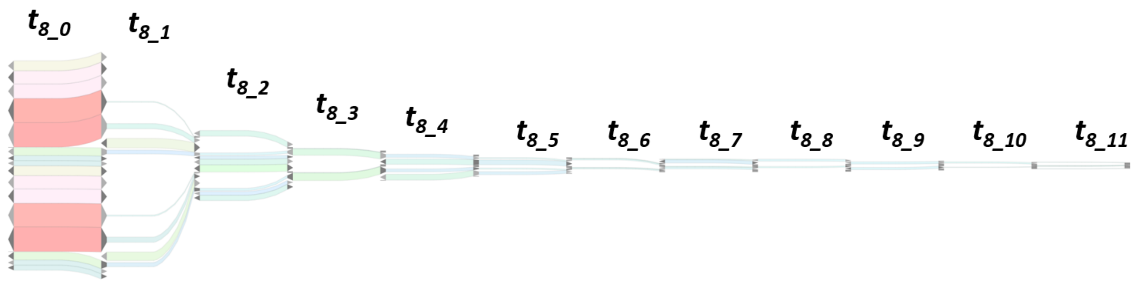


Figure 4.5: Visualization of 2-3 swap with asynchronized communication.

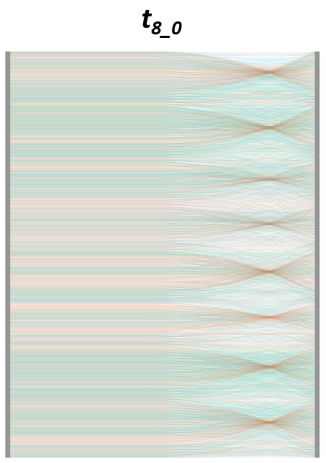


Figure 4.6: A zoom-in view of t_{8_0} in Figure 4.5 by displaying all 4320 processors.

occurred in the first stage. This justifies our understanding of the asynchronized communication behaviors of 2-3 swap.

4.2 AMR-based Simulations

We also apply our framework to Adaptive Mesh Refinement (AMR) based simulations. During the runtime of such a simulation, the block structured grids are adaptively refined over time, exhibiting dynamic and complex data exchanges across processors. To illustrate different communication activities, we consider two AMR-based appli-

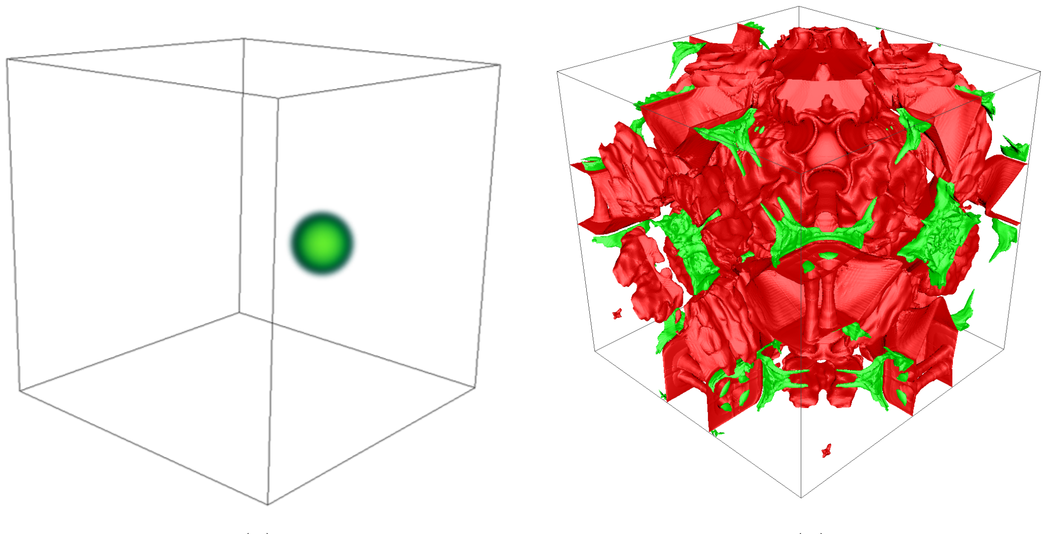
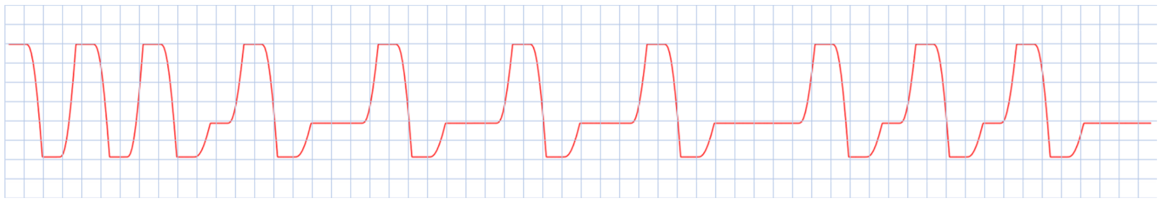


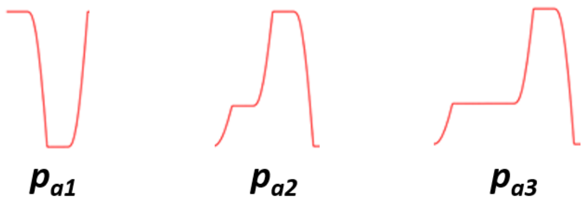
Figure 4.7: (a) and (b) show the renderings of the scalar data generated from the first and second AMR simulations on 4096 processors, respectively.

cations from the Chombo package [1] developed by the Lawrence Berkeley National Laboratory. The first application solves the advection-diffusion equation. The grid refinement of this application is relatively marginal over time, implying a possible stable communication among the processors. The second application solves the Euler equations of polytropic gas dynamics using integrating systems of conservation laws, incurring a higher level of grid refinement with an uneven spatial distribution and imbalanced data communication among the processors. Figure 4.7 shows the visualizations of the scalar data generated from these two simulations on 4096 processors. We can clearly see that the data set of polytropic gas simulation conveys more complex structures.

Figure 4.8 (a) shows the CommGram curve of the advection-diffusion simulation. The curve exhibits three main different communication patterns, p_{a1}, p_{a2}, and p_{a3}, as shown in Figure 4.8 (b). They periodically occur during the execution of the simulation. To examine the detailed communication activity of p_{a1}, p_{a2}, and p_{a3}, we



(a)



(b)

Figure 4.8: (a) shows the CommGram curve of the advection-diffusion simulation on 4,096 processors. (b) shows three different communication patterns, p_{a1}, p_{a2}, and p_{a3}, identified from the CommGram curve.

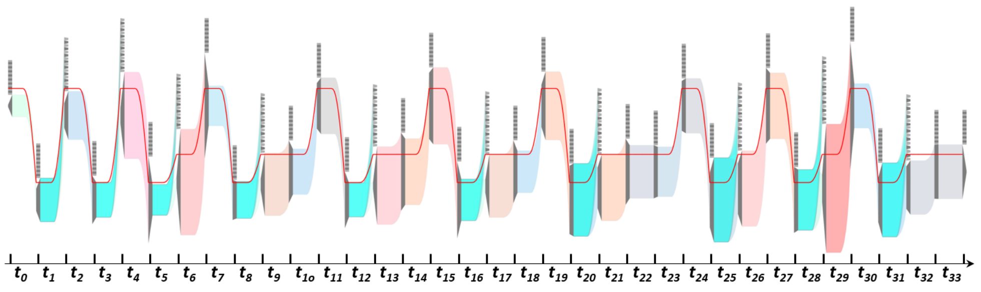


Figure 4.9: The polygon-base visualization of bipartite graphs shows the message flow of each pattern throughout the advection-diffusion simulation.

superimpose the polygon-based visualization of bipartite graphs over the CommGram curve, as shown in Figure 4.9, where message sizes are mapped from light blue to pink with light blue being the smallest size and pink being the largest size. The temporal and processor clustering methods are used to extract the communication phases and processor communities.

Figure 4.10 shows a view that magnifies the first four phases in Figure 4.9, corresponding to two instances of p_{a1}. In t_0, there is only one communication occurring

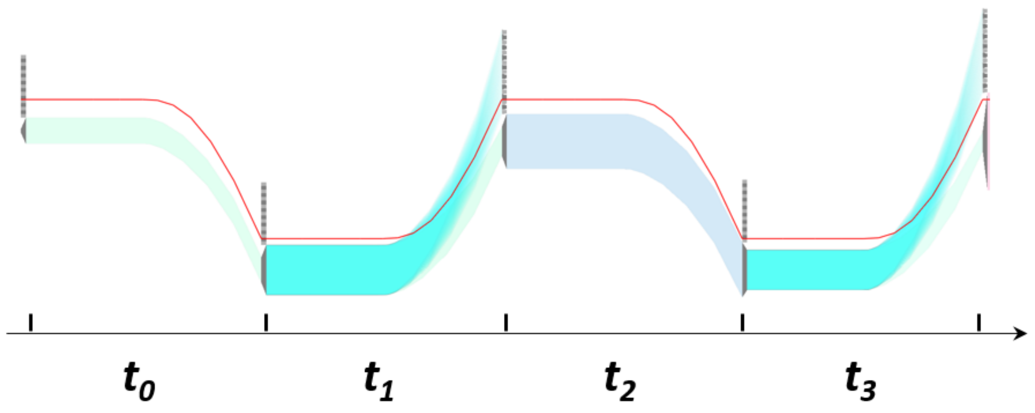


Figure 4.10: Magnifying the interval from t_0 to t_3 in Figure 4.9.

within one community, and there is no any inter-community communication. Then, in t_1, one community broadcasts messages to all other communities. The communication activity in t_0 and t_1 constitute p_{a1}. We can observe that the nearly identical activity is also repeated in t_2 and t_3, corresponding to another instance of p_{a1}.

Figure 4.11 magnifies the interval from t_5 to t_7 in Figure 4.9, corresponding to one instance of p_{a2}. Within this pattern, there is first a broadcast from one community in t_5, followed by an intra-community communication with a considerable large amount of message in t_6, and another intra-community communication with a less amount of message in t_7. This communication pattern also happens in the intervals from t_{25} to t_{27}, and from t_{28} to t_{30}. Similarly, we can also observe that p_{a3} corresponds to a nearly constant communication activity across several intervals.

Compared to the advection-diffusion simulation, the polytropic gas simulation has more complex structures and dynamic communication activity. Figure 4.12 (a) shows its CommGram curve. The beginning of the curve corresponds to the initialization stage of this simulation where the level of grid refinement is relatively high with an involvement of significant data exchanges. The communication exhibits more repet-

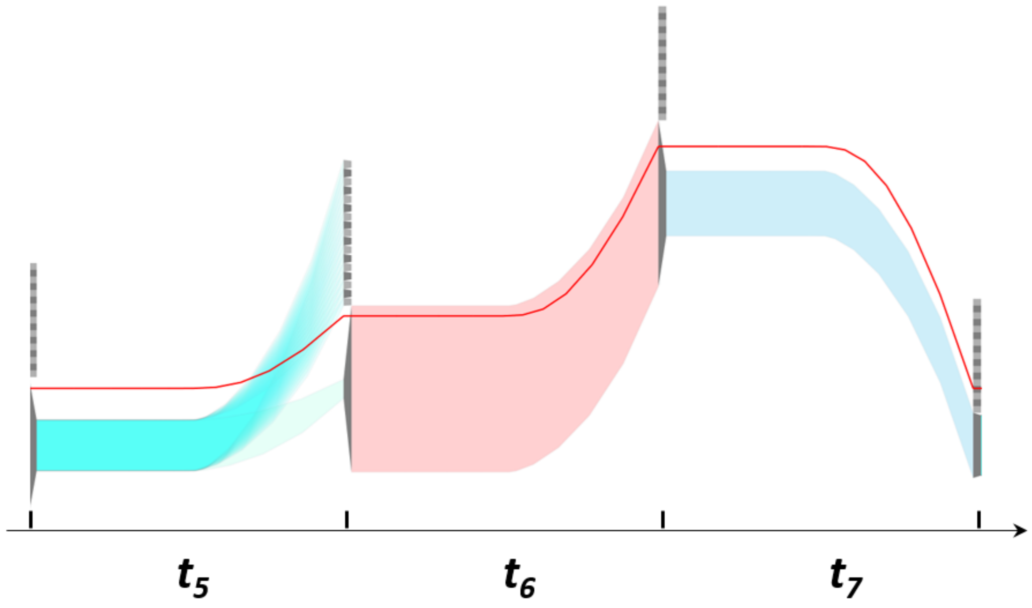
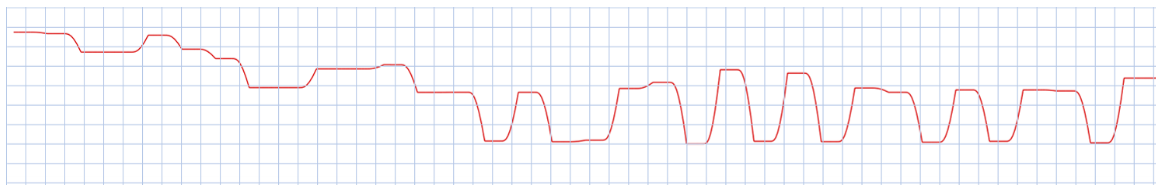
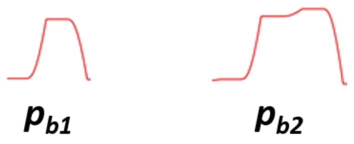


Figure 4.11: Magnifying the interval from t_5 to t_7 in Figure 4.9.

itive patterns when the grid refinement becomes more stable through the execution. Two major patterns, p_{b1} and p_{b2}, can be spotted in the CommGram curve, as shown in Figure 4.12 (b). Figure 4.13 shows more details using the polygon-based visualization. We can see that there are intensive intra-community communications in the beginning of the simulation, implying that grid refinement may be mainly conducted within a set of processors. From t_{14}, the communication activity tends to be more stable, and two patterns dominate the overall communication. We magnify the interval from t_{17} to t_{19}. In order to gain more precise information, our tool allows us to further zoom into one of the overlapping polygons incident to the first community of t_{19} and investigate the communication details. By leveraging the cluster tree generated from the processor clustering, the visualization displays more detailed communications within the selected polygon by choosing a finer LOD. As shown in Figure 4.13, an ALL_REDUCE communication is clearly depicted in the zoom-in



(a)



(b)

Figure 4.12: (a) shows the CommGram curve of the polytropic gas simulation on 4,096 processors. (b) shows two different communication patterns, p_{b1} and p_{b2}, identified from the CommGram curve.

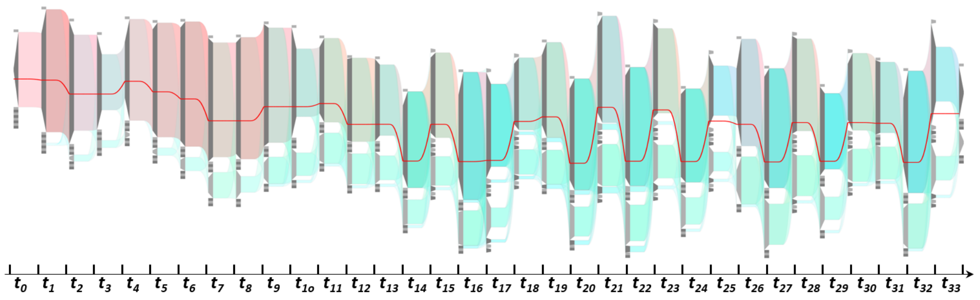


Figure 4.13: The polygon-base visualization of bipartite graphs shows the message flow of each pattern of the polytropic gas simulation.

view. We can continuously use zoom-in functionality on the selected polygon until reaching the finest LOD.

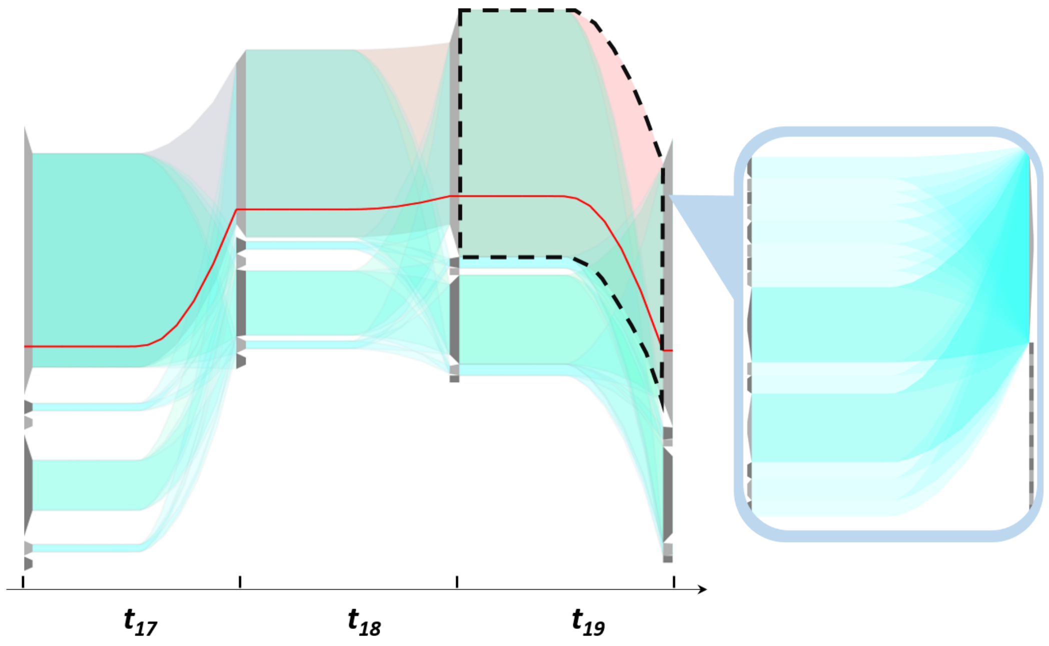


Figure 4.14: The left image shows a magnified view of the interval from t_{17} to t_{19} in Figure 4.13. The right image shows the detailed communication of the selected polygon. We can clearly see an operation of ALL_REDUCE.

Chapter 5

Conclusion

In this thesis, I present a novel visualization framework CommGram for communication analysis of parallel applications, and apply it to the end-to-end in-situ processing pipeline and AMR simulations. The result of my study provides an example illustrating the usage of CommGram. The CommGram fundamentally addresses the visual clutter and scalability issue by leveraging the hierarchical clustering methods. It can effectively characterize communication activity of large-scale parallel programs and provide a simple and concise representation that facilitates the monitoring and diagnosis of complex communication activity. Representative patterns and possible abnormalities can be identified through CommGram in an intuitive fashion. Besides the demonstrated studies, my framework can also work on other real-world applications and offer a feasible solution for analysts and researchers to explore communication behaviors and derive optimization strategies for large parallel programs.

In the future, I will continue to improve the framework and ensure its effectiveness to satisfy the increasing demand of performance analysis. The framework can be extended to visualize more run-time phenomena, such as bandwidth and network contention. I plan to add more visualization techniques, such as illustrative visualization, to enhance CommGram. I also intend to use the framework to study the data movement patterns within CPU cores and GPUs, and gain a deeper understanding of intra-node communication and its impacts on parallel processing.

Bibliography

[1] M. Adams, P. Colella, D. T. Graves, J.N. Johnson, N.D. Keen, T. J. Ligocki, D. F. Martin, P.W. McCorquodale, D. Modiano, P.O. Schwartz, T.D. Sternberg, and B. Van Straalen. Chombo software package for AMR applications - design document. Technical report, Lawrence Berkeley National Laboratory Technical Report LBNL-6616E.

[2] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[3] Robert Bell, AllenD. Malony, and Sameer Shende. ParaProf: A portable, extensible, and scalable tool for parallel performance profile analysis. In *Euro-Par 2003 Parallel Processing*, volume 2790 of *Lecture Notes in Computer Science*, pages 17–26. Springer Berlin Heidelberg, 2003.

[4] Abhinav Bhatele, Todd Gamblin, Katherine E. Isaacs, Brian T. N. Gunney, Martin Schulz, Peer-Timo Bremer, and Bernd 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*, SC '12, pages 31:1–31:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[5] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70:066111, Dec 2004.

[6] Antoni Bayes de Luna. *Clinical Electrocardiography: A Textbook, 4th Edition.* Wiley, 2012.

[7] Michael T. Heath and Jennifer Etheridge Finger. Paragraph: A tool for visualizing performance of parallel programs. Technical report, 1993.

[8] Gilbert Hendry and Arun Rodrigues. Sst: A simulator for exascale co-design. In *ASCR/ASC Exascale Research Conference*, 2012.

[9] Tobias Hilbrich, Joachim Protze, Martin Schulz, Bronis R. de Supinski, and Matthias 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*, SC '12, pages 30:1–30:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[10] Torsten Hoefler and Timo Schneider. Runtime detection and optimization of collective communication patterns. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12, pages 263–272, New York, NY, USA, 2012. ACM.

[11] Andreas Knüpfer, Ronny Brendel, Holger Brunst, Hartmut Mix, and WolfgangE. Nagel. Introducing the Open Trace Format (OTF). In VassilN. Alexandrov, GeertDick van Albada, PeterM.A. Sloot, and Jack Dongarra, editors, *Computational Science - ICCS 2006*, volume 3992 of *Lecture Notes in Computer Science*, pages 526–533. Springer Berlin Heidelberg, 2006.

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

[13] A.G. Landge, J.A. Levine, A. Bhatele, K.E. Isaacs, T. Gamblin, M. Schulz, S.H. 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*, 18(12):2467–2476, Dec 2012.

[14] Germán Llort, Harald Servat, Juan González, Judit Giménez, and Jesús 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*, SC '13, pages 29:1–29:11, New York, NY, USA, 2013. ACM.

[15] Glenn Luecke, Hua Chen, James Coyle, Jim Hoekstra, Marina Kraeva, and Yan Zou. MPI-CHECK: a tool for checking fortran 90 MPI programs. *Concurrency and Computation: Practice and Experience*, 15(2):93–100, 2003.

[16] Kwan-Liu Ma, James S. Painter, Charles D. Hansen, and Michael F. Krogh. Parallel volume rendering using binary-swap compositing. *IEEE Comput. Graph. Appl.*, 14(4):59–68, July 1994.

[17] Chris Muelder, Francois Gygi, and Kwan-Liu Ma. Visual analysis of inter-process communication for large-scale parallel computing. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1129–1136, November 2009.

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

[19] M.E.J. Newman. Detecting community structure in networks. *The European Physical Journal B - Condensed Matter and Complex Systems*, 38(2):321–330, 2004.

[20] Jorge Luis Ortega-Arjona. *Architectural Patterns for Parallel Programming: Models for Performance Estimation*. VDM Verlag, Germany, 2009.

[21] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76:036106, Sep 2007.

[22] Vivek Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, Cambridge, MA, USA, 1989.

[23] Sameer S. Shende and Allen D. Malony. The tau parallel performance system. *Int. J. High Perform. Comput. Appl.*, 20(2):287–311, May 2006.

[24] Carmen Sigovan, Chris Muelder, and Kwan-Liu Ma. Visualizing large-scale parallel communication traces using a particle animation technique. *Comput. Graph. Forum*, 32(3):141–150, 2013.

[25] Carmen Sigovan, Chris Muelder, Kwan-Liu Ma, Jason Cope, Kamil Iskra, and Robert 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*, IPDPS '13, pages 308–319, Washington, DC, USA, 2013. IEEE Computer Society.

[26] Yanhua Sun, Gengbin Zheng, Chao Mei, Eric J. Bohm, James C. Phillips, Laximant V. Kalé, and Terry 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*, SC '12, pages 55:1–55:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[27] Jeffrey S. Vetter and Bronis R. de Supinski. Dynamic software testing of MPI applications with Umpire. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, SC '00, Washington, DC, USA, 2000. IEEE Computer Society.

[28] Zuchao Wang, Min Lu, Xiaoru Yuan, Junping Zhang, and Huub van de Wetering. Visual traffic jam analysis based on trajectory data. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2159–2168, December 2013.

[29] Jieting Wu, Jianping Zeng, Hongfeng Yu, and Joseph P. Kenny. Commgram: A new visual analytics tool for large communication trace data. In *Proceedings of the First Workshop on Visual Performance Analysis*, VPA '14, pages 28–35, Piscataway, NJ, USA, 2014. IEEE Press.

[30] Hongfeng Yu, Chaoli Wang, and Kwan-Liu Ma. Massively parallel volume rendering using 2-3 swap image compositing. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 48:1–48:11, Piscataway, NJ, USA, 2008. IEEE Press.

[31] Omer Zaki, Ewing Lusk, William Gropp, and Deborah Swider. Toward scalable performance visualization with jumpshot. *Int. J. High Perform. Comput. Appl.*, 13(3):277–288, August 1999.