LBVis: Interactive Dynamic Load Balancing Visualization for Parallel Particle Tracing

Jiang Zhang[1] * Changhe Yang[1] † Yanda Li[1] ‡ Li Chen[3] § Xiaoru Yuan[1,2] ¶

1) Key Laboratory of Machine Perception (Ministry of Education), and School of EECS, Peking University, Beijing, China
2) Beijing Engineering Technology Research Center of Virtual Simulation and Visualization, Peking University, Beijing, China
3) Institute of CG&CAD, School of Software, Tsinghua University, Beijing, China

ABSTRACT

We propose an interactive visual analytical approach to exploring and diagnosing the dynamic load balance (data and task partition) process of parallel particle tracing in flow visualization. To understand the complex nature of the parallel processes, it is necessary to integrate the information of the behaviors and patterns of the computing processes, data changes and movements, task status and exchanges, and gain the insight of the relationships among them. In our proposed approach, the data and task behaviors are visualized through a graph with a fine-designed layout, in which node glyphs are dedicated to showing the status of processes and the links represent the data or task transfer between different computation rounds and processes. User interactions are supported to facilitate the exploration of performance analysis. We provide a case study to demonstrate that the proposed approach enables users to identify the bottlenecks during this process, and thus help optimize the related algorithms.

Keywords: Parallel particle tracing, load balancing, performance analysis, visual analytics

1 INTRODUCTION

Parallel particle tracing is a fundamental technique in flow visualization. In common methods, by partitioning the flow data into blocks and distributing data blocks and particle seeds over different processes, users can trace particles to conduct many applications. The workload of a process is defined by the total integral steps of particles calculated by this process. As the advection of particles is always considered unpredictable, a key problem in parallel particle tracing is load balancing. It is very likely that the workload distribution over parallel processes is imbalanced because the initial assignments are not even. Many algorithms have been proposed to address this problem. Typical methods include transferring jobs from busy processes to idle processes [6] and performing dynamic data repartitioning to reassign data blocks to processes with equal workload [21]. These methods optimize load balancing from an algorithmic level by specifically processing and scheduling data and tasks. However, the dynamic process of load-balanced parallel particle tracing is still implicit.

We focus on exploring the performance data related to load-balancing from the perspective of visualization and visual analysis. During parallel particle tracing, it is feasible to obtain the statistics or even track the movement of each data block and the workload changes of each process in the whole process. The obtained information is closely related to the efficiency of corresponding the load-balancing algorithms. Visualization allows us to better understand this information in an intuitive and interactive way. By visually analyzing the intermediate behaviors of parallel processes, we can flexibly and efficiently understand the data and task exchange among processes at different stages of run time. Moreover, analyzing the performance information also helps to choose better parameters or customize suitable load-balancing algorithms to purposefully improve the performance of parallel particle tracing applications.

Currently, researchers for analyzing the performance in parallel applications focus on exploring the communication patterns. Examples include investigating the traffic congestion [3] and optimizing the routing scheme [7]. These methods only consider the quantities of the messages that are communicated among computing nodes. In large-scale flow visualization, however, the dynamic distribution and transfer of data blocks and tasks during run time are the focus of load-balancing process, from which we can analyze the characteristics of related applications and possible bottlenecks of load imbalance. Traditional performance profiling tools [1, 16] are not applicable to obtain the performance information and analyze them efficiently. There is little work targeting this challenging problem.

In this work, we present a novel visual analytical approach to analyze the dynamic process of load balancing using the intermediate performance data dumped during the run-time parallel particle tracing for parallel algorithm diagnostic experts. In our design, a node-link graph is used to visualize the behaviors of each process, the changes and movements of data blocks, and the status and exchange of tasks. With a well-designed layout, the node glyphs are dedicated to showing the behaviors of processes, while the links represent the data or task transfer between different computation rounds among processes. We also delicately design the encodings for the distribution of data blocks assigned to each process and allow interactively tracking the transfer path of a data block to explore the patterns related to data block movement. Interactive operations including ranking based on different metrics and reordering are supported for users to flexibly explore the important processes.

With multiple visualization and interaction designs, we apply the proposed approach to an existing dynamic load-balancing algorithm for parallel particle tracing in flow visualization and optimize it using the visual discoveries. The case study demonstrates the effectiveness of our method.

2 RELATED WORK

In this section, we present the background by reviewing related work on load balancing in parallel particle tracing and performance visualization and analysis.

2.1 Load Balancing in Parallel Particle Tracing

Researches on load balancing in parallel particle tracing are critical for large-scale flow visualization. No matter in data-parallel or in task-parallel methods that are categorized by seeding strategies, load balancing is a key problem.

*e-mail: jiang.zhang@pku.edu.cn
†e-mail: 2014ych@pku.edu.cn
‡e-mail: yanda_li@pku.edu.cn
§e-mail: chenlee@tsinghua.edu.cn
¶e-mail: xiaoru.yuan@pku.edu.cn (corresponding author)

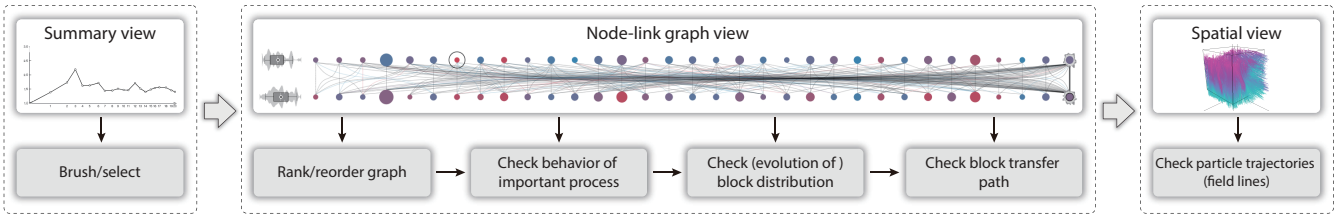

Figure 1: Visual exploration pipeline of our method. Flexible interactions are provided to analyze the patterns of important processes and the relationships among important processes.

In data-parallel methods, data is statically partitioned into blocks and then assigned to different processes. In order to improve load balance, both static and dynamic load-balancing algorithms are proposed. Static load-balancing algorithms attempted to average the workload of different processes through preprocessing before run time. Examples include partitioning data based on static workload estimation [14] and flow direction [4], and assigning data according to round-robin strategy [15]. Dynamic load-balancing algorithms periodically perform workload redistribution by data movement during run time. A representative method is dynamically repartitioning the data based on the estimation of block workload for the tracing in the immediate future [21].

In task-parallel methods, particles are statically assigned to different processes. Algorithms such as work stealing [6] and work requesting [12] are proposed to dynamically improve the load balance.

Above methods address the problem of load balancing from the algorithm level. In this work, we propose a visual analysis approach to interactively analyze and explore the root cause of load imbalance in data-parallel particle tracing algorithms from a micro perspective.

2.2 Performance Visualization and Analysis

Performance visualization is an important method to analyze the characteristics of large-scale parallel applications. Researchers have developed several general-purpose performance profiling and visualization tools, such as Jumpshot [20], ParaGraph [8], and Vampir [13]. For specific parallel applications, it requires more scalable visualization methods.

Boxfish [9] uses 3D mesh representation or 2D projection to visualize 3D torus networks. The projection method in Boxfish is also extended for the visualization of 5D torus networks [10]. Ring layout [2] and TorusVisND [5] that combines ring layout and parallel coordinates are also developed for high-dimensional torus network. The communication patterns are also important in affecting the efficiency of parallel applications. Wu et al. [18] use temporal clustering and processor clustering for the level-of-detail exploration of communication trace. Bhatia et al. [3] design an interactive web-based visualization tool called TreeScope to investigate traffic congestion on fat-tree networks. Recently, a call stack tree structure is built to represent the executions of high performance computing [19]. To analyze and further optimize the communication routes, Fujiwara et al. [7] proposed a visual analytic system to identify the communication bottlenecks in parallel MPI programs. The representative MPI calls can also be visualized by plotting the start/end times and duration [11], or using a particle animation technique [17].

Different from existing work, we concentrate on the load-balancing problem in parallel particle tracing and analyze the intermediate performance information by several targeted visualization and interaction designs.

3 Overview

In the following, we first interpret the design requirements that feature the targeting problem, and then we introduce the analytical data and tasks that our visualization design should address.

3.1 Design Requirements

As introduced before, the process of dynamic load balancing in parallel particle tracing is implicit. The algorithm accepts specified input parameters and outputs a load-balancing run. Researchers always focus on the running result about whether the performance is improved or not, but it is difficult for them to visualize the intermediate run-time process that brings insightful information. From the perspective of visual analysis, it is critical to explore the dynamic process, in order to understand how a load-balancing algorithm balances the workload through data block movement and task exchange. Therefore, the most important requirement in our visualization design is enabling visually analyzing the algorithm characteristics according to the intermediate performance information. The advantages and disadvantages of these algorithms with respect to the transfer patterns of data blocks and tasks should be revealed by the visualization and visual exploration.

According to discussions above, we developed two categories of requirements that our visualization design should support:

- **R1: Pattern exploration** Explore the patterns on important processes and the behaviors of their assigned data blocks and tasks;
- **R2: Relationship exploration** Explore the relationships among important processes according to the transfer of data blocks and tasks.

3.2 Data and Task Description

Here we introduce the analytical data and tasks that correspond to the design requirements. The analytical data are collected from the intermediate performance information related to load balancing during run-time parallel particle tracing. We follow the workflow of general load-balancing algorithms that are employed in block-synchronous parallel particle tracing [14, 21]. In these methods, the whole process could be managed into rounds. Processes trace particles independently in each round, then exchange particles and data blocks after each round of particle tracing.

In this work, we mainly focus on the dynamic load-balancing algorithms that are performed through the transfer of data blocks and tasks. We embed profiling codes in this dynamic process to dump related performance information for visualization and visual analysis. Two types of performance data are obtained in each round. The first one is the distribution of data blocks over processes and tasks over blocks. The other is related to the source/destination processes of the incoming/outgoing data blocks and the source/destination blocks of the incoming/outgoing tasks.

Based on the performance data derived from the dynamic load-balancing process, we developed several analytical tasks in our work:

- **T1:** Show spatial information according to the distribution of particles/blocks in the domain.
- **T2:** Show the overview of load balancing and the outlying rounds with unbalanced workload by the temporal statistics.
- **T3:** Show the workload and block distribution over processes as well as the relationships among them in detail.

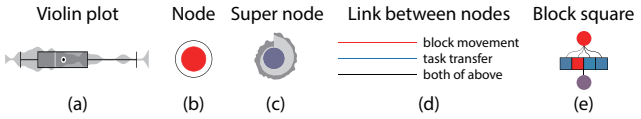

Figure 2: Glyph and icons in the node-link graph view. A node glyph represents a process, and squares represent data blocks.

- **T4:** Explore the patterns in important processes with their blocks to analyze the load balancing.
- **T5:** Make comparison to explore the relationships between important processes from block level.

Above analytical tasks are closely related to the design requirements described in Section 3.1. To fulfill these tasks, we need to develop visualizations and interactive operations to explore the patterns of processes and the relationships among different processes.

4 Visualization with Interactive Exploration

Our visualization interface consists of three views, including the summary view (**T2**), node-link graph view (**T3**), and spatial view (**T1**). The three views are linked together through interactions to explore from overview to details (**T4** and **T5**), as shown in Figure 1.

Summary View

We utilize a line chart to display the overview of a load-balancing process. The horizontal axis represents the execution time, and we mark the number of rounds on it. The vertical axis represents the load-balancing indicator, which is computed as the ratio of the maximal process workload to the average workload of processes in this round. We plot the load-balancing indicator at each time when a round of tracing is complete. Note that the horizontal axis is not evenly spaced because the complete time of each round is different. From this view, we can easily capture the trend of load balancing over time and find abnormal patterns (i.e., significant load imbalance) in certain rounds. We also provide brushing and linking interactions to help users focus on these important rounds.

Node-Link Graph View

We provide a node-link graph as the main view to visualize behaviors of processes and the transfer of data blocks and tasks among processes. The graph structure uses general graph properties including node glyph and links.

Node A node glyph represents a process. The size of the node encodes the total number of data blocks distributed in this process in the corresponding round. A color scheme is applied to encode the workload of each process. The color gradually changes from blue to red, indicating that the workload linearly changes from light to heavy. Especially, we add an outer circle around the nodes to highlight the processes with the heaviest workload, as shown in Figure 2(b). Users can hover on the corresponding node to show the detailed information, including the workload and the number of assigned blocks and particles. By clicking on a node in a certain round, the nodes and links associated are highlighted.

Due to the limited space in the screen, the number of nodes displayed on the interface is constrained. In consideration of the scalability of our design, we only show a specific number of processes that are of higher significance for each round. The rest of the processes are organized as a **super node** glyph, as shown in Figure 2(c). The super node is a node surrounded by a radial stacked area chart. We encode the node with statistical summary of the rest of the processes. That is, the color of the node indicates the average workload, while the node size represents the average number of assigned blocks. The radial stacked area chart in the super node has two layers. The inner layer shows the workload distribution over the rest of the processes, while the outer layer shows the distribution of the number of blocks. Note that the color in the stacked areas has no special meaning but to differentiate the two layers. The number of individually-shown processes can be specified by users.

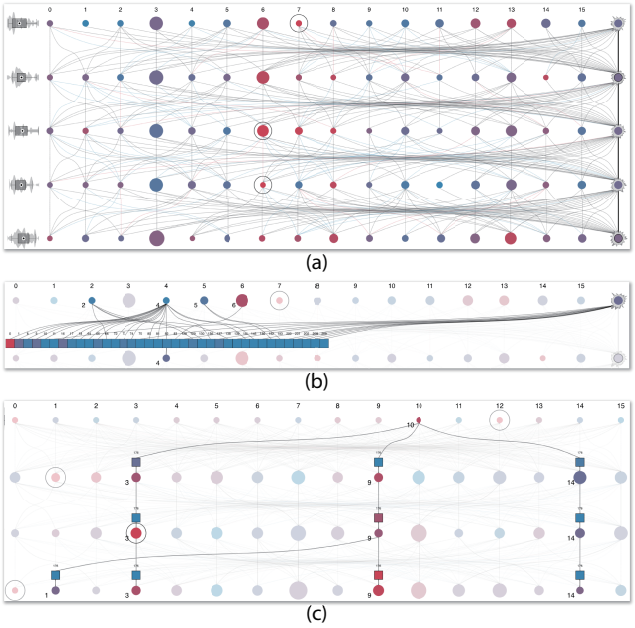

Figure 3: (a) Node-link graph view showing the behaviors of each process and the transfer of data blocks and tasks among processes; (b) Distribution of data blocks represented by squares; (c) Transfer path of a block over rounds.

Link A link between a pair of nodes represents the movement of data blocks or exchange of tasks from one process in a round to another in the next round. For the super node, the connected links indicate data blocks or tasks are transferred from/to the rest of the processes. We use a curve between the two corresponding nodes to encode the link. Color encoding is also employed to display different types of link. As illustrated in Figure 2(d), a link in red indicates data block movement, while a blue link represents the exchange of tasks. When data block movement and task exchange both exist, the link is colored in black. The width of links represents the number of moved data blocks or exchanged tasks.

The nodes and links are arranged with a layout as shown in Figure 3(a). The graph view displays several rows of nodes from top to bottom, indicating consecutive rounds of tracing. Nodes in each column indicate the same processes in different rounds. With this layout, the distribution over processes in each round is shown horizontally, and we can also easily see the behavior change of a process over rounds by vertically comparing the nodes. We also provide ranking operations to rearrange the order of each column of nodes to emphasize the important processes. After ranking, the displayed nodes may partially or all come from the original super node, and the rest of the nodes are re-summarized as a new super node.

We add a **violin plot** (Figure 2(a)) that is placed in front of the nodes to capture the overall characteristics of workload distribution. The violin plot is the combination of the box plot and kernel density plot, which summarizes the workload distribution over processes in each round. In the kernel density plot, the horizontal direction represents the amount of workload from light (left) to heavy (right), while the vertical direction indicates the density of processes. The overlaid box plot shows 5% percentile, quartiles, and 95% percentile of the distribution.

Apart from visualizing the behaviors of processes by the node-link glyph, we also encode the data blocks assigned to each process to show detailed information. Each data block is visualized as a **square** (Figure 2(e)). For each process, the squares of its assigned data blocks are displayed horizontally over the corresponding node, as shown in Figure 3(b). The color scheme for the block squares is

the same as we used in node encoding. The squares are connected with the nodes in the previous round, which indicates that their associated data blocks come from those nodes. By clicking on a square, the transfer path of the corresponding data blocks over all rounds (i.e., all the nodes traveled by this block with the links showing the movement) is highlighted, as shown in Figure 3(c).

Our visualization also supports comparing the distribution of blocks over processes in different rounds. An operation to align the block distribution of a process over rounds or align the block distribution of different processes is provided. Blocks can be aligned vertically, and a vertical line appears to indicate which blocks are the same ones when hovering on the block square.

Spatial View

We present a spatial view to check the particle trajectories (i.e., field lines) when selecting a data block. The workload of a block is directly related to the tracing tasks in it, and it is only encoded by colors in the node-link graph view. The spatial view is then used to show the distribution of block workload and to further help validate the status of tasks in the corresponding blocks. Users can check the field lines generated in the important blocks, compare the task status in different blocks, and identify the cause of abnormal workloads.

5 Case Study

In this section, we present a case study that purposefully optimizes an existing dynamic load-balancing algorithm according to the visual findings from our approach.

We perform the dynamic load-balancing algorithm on Tianhe-2 supercomputer[1] and collect the related profiling data. This algorithm periodically performs data repartitioning and then reassign data blocks according to repartitioning results after each round of tracing [21]. Data duplication in different processes is allowed when the particles can travel through multiple other blocks other than the originating blocks in each round. Two flow data are used for parallel particle tracing in the experiments, including thermal hydraulics simulation (Nek5000) data and hurricane Isabel data. The Nek5000 data is generated from the Nek5000 solver, which comes from a large-eddy Navier-Stokes simulation at Argonne National Laboratory. A single time step of the data with a spatial resolution of $512 \times 512 \times 512$ is used for streamline computation. The Isabel data comes from an atmospheric simulation, which is developed by the National Center for Atmospheric Research in the United States. This data has a spatial resolution of $500 \times 500 \times 100$ and we also use a single time step for the experiment.

We first visualize the performance data after running the dynamic load-balancing algorithm with the Nek5000 data. Figure 4(a) shows the summary view of the evolution of load-balancing indicator for 32 processes. From this view, we can find that a peak appears in the third round which indicates the occurrence of load imbalance. We want to explore the cause of abnormality in detail. We brush the first five rounds next to this round and check the graph view, as shown in Figure 4(b). The violin plot of the third round indicates that there are few processes with much heavier workloads. We rank the processes according to their workloads. It can be found that process 24 is with the heaviest workload in this round (Figure 4(c)).

Furthermore, in order to check the behavior of process 24, we display the block distribution in it and align the blocks of this process, as shown in Figure 4(d). The visualization of block squares shows that block 224 has much heavier workload than that of other blocks. By clicking on block 224, the transfer paths of this block are shown. Note that a block can be distributed to different processes with multiple duplications in each round. From the transfer paths of block 224 in Figure 4(e), we find that there are always certain block duplications that appear in red in each round, which implies the workloads of these duplications are always heavier. In order to check the root cause, we render the particle trajectories (i.e., streamlines) generated in block 224 and other blocks in the third round, as shown in Figure 5. The spatial views validate that block 224 has much more tracing tasks, which leads to heavier workload. Blocks with extremely heavy workloads also cause heavy workloads in the corresponding processes. During the transfers of these blocks, processes in different rounds are "infected" and have increased workloads. This also builds relationships among processes.

Above findings demonstrate the data-repartitioning algorithm did not perform well when the workload distribution on blocks is uneven. The related algorithm takes each block as a unique reparti-

[1] https://www.top500.org/featured/systems/tianhe-2/

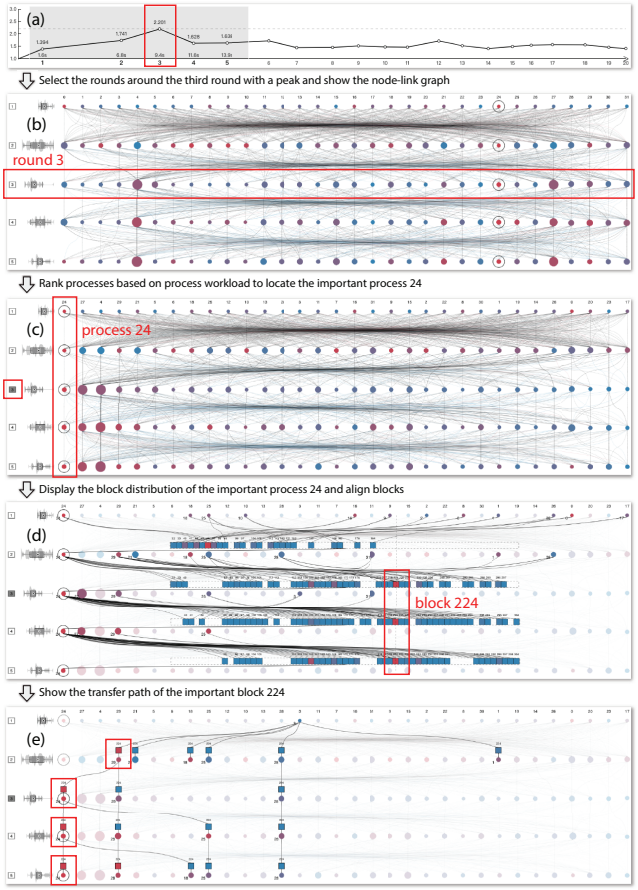

Figure 4: A visual exploration case for analyzing the bottleneck of the data-repartitioning algorithm with the Nek5000 data. Through interactive exploration, it reveals that block 224 always leads to heavy workload of corresponding processes.

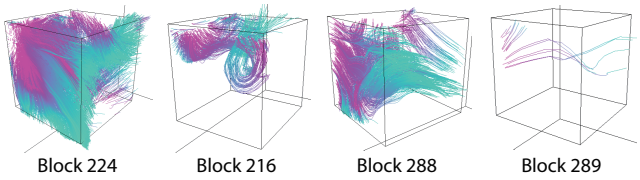

Figure 5: Spatial views of the Nek5000 data in different data blocks assigned to process 24 in the third round. The color of field lines encodes integration steps during particle tracing. The workload in block 224 is significantly more than that in others.

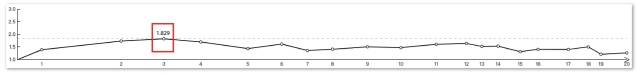

Figure 6: The summary view generated with the performance data collected from the optimized algorithm for parallel particle tracing with the Nek5000 data.

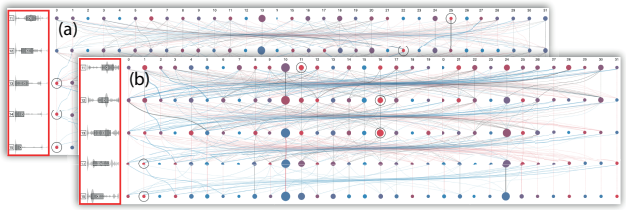

Figure 7: The node-link graph from the eleventh round to the fifteenth round for (a) the data-repartitioning algorithm and (b) the optimized algorithm with the Isabel data.

tioning object with equal status. But in fact, it should be better to have special treatments for blocks with much heavier workload.

With these findings, researchers could further optimize the related algorithm in a targeted manner because the algorithm limitations have been visually revealed. Specifically, as there are always blocks like block 224 with heavier workload than other blocks in certain rounds, each of these blocks can be organized into two copies, and each copy has half of the workload. The two copies replace the original one as two independent repartitioning objects during data repartitioning. This optimization makes the workload distribution of data blocks more even and thus obtains more balanced repartitioning. Figure 6 shows the summary view of the optimized result. Compared with the original algorithm, we can find that the load-balancing indicator of the third round in the optimized one is lower. After optimizing the algorithm, the overall load-balancing indicator is reduced by 8.7%.

We further use the Isabel data to test the algorithm after optimization. After visualizing the collected performance data in the views, it shows that the load balancing is also improved. In the original algorithm, the load becomes significant imbalance, as shown in the violin plot of Figure 7(a) from the eleventh round to the fifteenth round. This is because the unfinished tasks are being localized and only few blocks are allocated with heavy workloads. The optimized algorithm can alleviate this problem. As shown in Figure 7(b), workload distribution over processes is more even, which can be found more obviously from the violin plot.

6 Conclusion

In this work, we proposed a visual analytical method for analyzing the intermediate information generated during the load-balancing process in parallel particle tracing. With dedicated visualization and interaction designs, we display the behaviors of the computing processes through a node-link graph structure, and explore the patterns on data movements and task exchanges during run time. According to the findings by visual analysis, our method can help to optimize the related algorithms. In the future, we would like to extend our method to support even larger-scale parallel processes with scalability.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. This work is supported by National Numerical Windtunnel Project and NSFC No. 61672055.

References

[1] L. Adhianto, S. Banerjee, M. W. Fagan, M. Krentel, G. Marin, J. M. Mellor-Crummey, and N. R. Tallent. HPCTOOLKIT: tools for performance analysis of optimized parallel programs. Concurrency and Computation: Practice and Experience, 22(6):685–701, 2010.

[2] A. Bhatele, T. Gamblin, K. E. Isaacs, B. T. N. Gunney, M. Schulz, P. Bremer, and B. Hamann. Novel views of performance data to analyze large-scale adaptive applications. In SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, Salt Lake City, UT, USA - November 11 - 15, 2012, p. 31, 2012.

[3] H. Bhatia, N. Jain, A. Bhatele, Y. Livnat, J. Domke, V. Pascucci, and P. Bremer. Interactive investigation of traffic congestion on fat-tree networks using treescope. Comput. Graph. Forum, 37(3):561–572, 2018.

[4] L. Chen and I. Fujishiro. Optimizing parallel performance of streamline visualization for large distributed flow datasets. In Proc. of IEEE Pacific Visualization Symposium 2008, pp. 87–94, 2008.

[5] S. Cheng, P. De, S. H. Jiang, and K. Mueller. Torusvisnd: unraveling high-dimensional torus networks for network traffic visualizations. In Proc. of the First Workshop on Visual Performance Analysis, VPA '14, New Orleans, Louisiana, USA, November 16-21, 2014, pp. 9–16, 2014.

[6] J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha. Scalable work stealing. In SC'09: Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 53:1–53:11, 2009.

[7] T. Fujiwara, P. Malakar, K. Reda, V. Vishwanath, M. E. Papka, and K.-L. Ma. A visual analytics system for optimizing communications in massively parallel applications. In Proc. of IEEE Conference on Visual Analytics Science and Technology (VAST'17), pp. 1–12, October 2017.

[8] M. T. Heath. Paragraph: A tool for visualizing performance of parallel programs. In Second Workshop on Environments and Tools for Parallel Scientific Computing, pp. 221–230, 1994.

[9] K. E. Isaacs, A. G. Landge, T. Gamblin, P. Bremer, V. Pascucci, and B. Hamann. Abstract: Exploring performance data with boxfish. In 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, USA, November 10-16, 2012, pp. 1380–1381, 2012.

[10] C. M. McCarthy, K. E. Isaacs, A. Bhatele, P. Bremer, and B. Hamann. Visualizing the five-dimensional torus network of the IBM blue gene/q. In Proc. of the First Workshop on Visual Performance Analysis, VPA '14, New Orleans, Louisiana, USA, November 16-21, 2014, pp. 24–27, 2014.

[11] C. Muelder, F. Gygi, and K. Ma. Visual analysis of inter-process communication for large-scale parallel computing. IEEE Trans. Vis. Comput. Graph., 15(6):1129–1136, 2009.

[12] C. Müller, D. Camp, B. Hentschel, and C. Garth. Distributed parallel particle advection using work requesting. In Proc. of IEEE Symposium on Large Data Analysis and Visualization 2013, pp. 1–6, 2013.

[13] W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach. Vampir: Visualization and analysis of mpi resources. Supercomputer, 12:69–80, 1996.

[14] B. Nouanesengsy, T.-Y. Lee, and H.-W. Shen. Load-balanced parallel streamline generation on large scale vector fields. IEEE Trans. Vis. Comput. Graph., 17(12):1785–1794, 2011.

[15] T. Peterka, R. B. Ross, B. Nouanesengsy, T.-Y. Lee, H.-W. Shen, W. Kendall, and J. Huang. A study of parallel particle tracing for steady-state and time-varying flow fields. In IPDPS'11: Proc. of IEEE International Symposium on Parallel and Distributed Processing, pp. 580–591, 2011.

[16] S. Shende and A. D. Malony. The tau parallel performance system. IJHPCA, 20(2):287–311, 2006.

[17] C. Sigovan, C. Muelder, and K. Ma. Visualizing large-scale parallel communication traces using a particle animation technique. Comput. Graph. Forum, 32(3):141–150, 2013.

[18] J. Wu, J. Zeng, H. Yu, and J. P. Kenny. Commgram: a new visual analytics tool for large communication trace data. In VPA'14: Proc. of the First Workshop on Visual Performance Analysis, pp. 28–35, 2014.

[19] C. Xie, W. Xu, and K. Mueller. A visual analytics framework for the detection of anomalous call stack trees in high performance computing applications. IEEE Trans. Vis. Comput. Graph., 25(1), 2019.

[20] O. Zaki, E. L. Lusk, W. Gropp, and D. Swider. Toward scalable performance visualization with jumpshot. IJHPCA, 13(3):277–288, 1999.

[21] J. Zhang, H. Guo, X. Yuan, and T. Peterka. Dynamic data repartitioning for load-balanced parallel particle tracing. In Proc. of IEEE Pacific Visualization Symposium 2018, pp. 86–95, 2018.