Characterizing Communication Patterns of Parallel Programs Through Graph Visualization and Analysis

Denise Stringhini(✉) and Alvaro Fazenda

Universidade Federal de Sao Paulo, Sao Paulo, Brazil
dstringhini@unifesp.br

Abstract. Characterization of communication patterns of parallel programs has been used to better understand the behavior of such programs as well as to predict performance of large scale applications. This characterization could be performed by observing some communication attributes like volume or spatial characteristics of message passing parallel applications in different scenarios. This paper describes a methodology to characterize parallel communication patterns using a graph visualization tool in addition to a traditional monitoring tool that generates trace files. Graph visualization tools are commonly used to analyze large network connections existent in a variety of social or natural structures. Although, since it is possible to represent large scale parallel programs as graphs of communicating processes, this paper proposes a methodology that takes advantage of such kind of tool to aid in characterize communication patterns.

1 Introduction

Basically, characterization of communication patterns in message passing parallel programs resides in to explore mainly three attributes of message passing programs: spatial distribution, volume of messages and temporal distribution. The spatial behavior is characterized by the distribution of messages destinations. The volume of data transferred is characterized by the distribution of message sizes and the average number of messages. The temporal behavior is characterized by the distribution of message generation rate [12].

The importance of such characterization relies in a better understanding of communication performance which has a crucial influence on the overall performance of a parallel program. A proper understanding of communication behavior of parallel applications may support the design of better communication subsystems as well as help application developers to maximize their application performance on a target architecture [21]. Usually, the methodology to characterize communication patterns consists in to dynamically record communication events and statistically analyze and organize those data post-mortem. The characterization data are commonly presented through bar graphs or tables. An example of the characterization of communication patterns to improve the performance of MPI [17] programs could be found in [15].

S. Hunold et al. (Eds.): Euro-Par 2015 Workshops, LNCS 9523, pp. 565–576, 2015.
DOI: 10.1007/978-3-319-27308-2_46

This paper proposes an alternative perspective to characterize communication patterns in large scale parallel applications. The methodology consists in to analyze the data recorded in a trace file by using a graph visualization tool and some metrics based on complex networks theory. In order to do this task, we first create a communication graph from an adjacency matrix. In this work, this graph is extracted directly from the communication matrix generated by EZTrace [18], a trace file generator. Second, the generated graph is loaded into the Gephi graph visualization tool [3]. It includes several graph layout algorithms to provide different views, as well as it provides some complex networks related metrics that are also used in the characterization.

In order to construct a suitable visualization it is important to choose the best layout algorithm as well as its parameters. In addition, tools like Gephi include a variety of filters which help in highlight some specific features of communication links. Our methodology defines some applicable layout choices in order to develop a relevant communication characterization. Specifically in this work, we use a determined layout and some complex networks metrics to preliminarily characterize the spatial and volume attributes of NAS parallel benchmarks [1]. The goal is to primary explore the methodology to demonstrate its potential.

As first results, the graph visualization and analysis allowed a topology characterization that is not easily obtained by state-of-art parallel visualization tools. After some tests with several layout options, it was possible to actually visualize the toroidal stencil shape of SP-MZ NAS parallel benchmark as well as to analyze its behavior as it scales. This kind of characterization could be very useful in the development phase of parallel applications.

Besides the benefits of the characterization itself, this proposal has the advantage of using existing tools. As in [11], we also claim that this approach avoids the high costs of building totally new visualization systems. The present work preliminarily expose the potential of the technique while we develop more specific ways to apply it to parallel systems by adding new information and temporal behavior.

This paper is organized as follows. First it is presented some related work and tools used in the methodology. After, it is described the methodology and it is presented a case study along with primary conclusions.

2 Related Work

Several works on characterization of communication patterns in parallel programs have been proposed over the years that explore mainly three attributes of message passing programs: temporal, spatial and volume. Besides, there are other attributes like the communication locality, explored in [12].

In [21] both point-to-point and collective communications are considered. For point-to-point they quantify the message type, message frequency, message size, and message destinations. For collectives, they examine their type, frequency, and payload. Their results show applications studied are sensitive to changes in the system size and the problem size. Lee in [14] also measured the communication timing, sources and destinations, and message size distributions in order

to characterize MPI programs. His experimental results also show such metrics could be used to predict performance.

In [19] a set of applications is characterized along four dimensions: point-to-point communication, collective communication, memory load operations, and floating point operations. Particularly for point-to-point communication, they measure distributions for number of messages, type, payload size, and size of destination clique.

This work presents a different methodology based on graph visualization and complex network theory. The first characterization provided is a topological view of the parallel application, that was not provided by any of the above mentioned related work. The first results show that is possible to easily visualize processes connections that exhibit some regularity in communication patterns and observe their behavior as the program scales. The topological view could be further explored by applying filters as the graph visualization tools are mostly interactive. Also, some common metrics from complex networks theory were used to characterize the applications. Metrics like the average degree, graph density and others were used to characterize the scalability of analyzed applications.

Besides previous work on characterization, it is also important to highlight some visualization tools as well as recent work on parallel program visualization, since in the present proposal graph visualization is key to characterization.

Vampir [13] is a parallel program visualization tool that provides a framework for program analysis, which enables developers to display program behavior at any level of detail. Performance data obtained from a parallel program execution can be analyzed with a collection of different performance views. The present work has no intention to repeat the level of detail provided by Vampir, but to present a complementary perspective.

The work in [16] presents another way to evaluate parallel programs and could be related to Vampir as another potential view. The work is related to the present effort in the sense that it allows the visualization of large parallel programs along with communication patterns. However, while their work is more focused on hardware topology, this work addresses the application topology.

Before to present the methodology, we briefly approach some basic complex network theory that are important to the characterization.

3 Complex Networks Basics

Graphs become increasingly important in modeling complicated and/or complex structures, such as circuits, images, chemical compounds, protein structures, biological networks, social networks, the Web, workflows, and even XML documents [8]. The research in graph mining and graph visualization is producing a large collection of techniques and tools that could be useful in a variety of research areas. This section explains how a large parallel message passing program can be modeled as a graph of communicating processes as well as how complex networks theory could help in characterizing communication patterns.

Recently, it can be observed a new movement of interest and research in the study of complex networks, i.e. networks whose structure is irregular, complex

and dynamically evolving in time, with the main focus moving from the analysis of small networks to that of systems with thousands or millions of nodes [4]. The complex network theory is aimed mostly to analyze very large and dynamical irregular networks, but some metrics and tools could also apply to regular structures like the ones formed by processes in a parallel program.

A graph G, is a pair of sets (V, E), where V is a finite set of vertices and E is a set of edges, each edge connecting a pair of vertices. In directed graphs (digraphs), each edge has a direction and self-loops are allowed. In undirected graphs, each edge is an unordered pair of vertices, thus the adjacency is symmetric. In weighted graphs, each edge has an associated weight, which is a value assigned to the edge.

A large parallel program consists of hundreds or thousands of distributed processes. These processes could be naturally modeled as vertices of a graph. In order to execute a parallel algorithm, processes have to exchange messages, which creates connections or links between pairs of processes. These communication operations could be modeled as the edges of a graph. The number or the volume of messages exchanged could be assigned to the edges as their weight. Considering these characteristics it is straightforward to think in a parallel program as a graph whose attributes could characterize its communication patterns.

More information about complex networks theory can be found in [4,7]. In this work we are particularly interested in metrics that are summarized below:

- **Average degree:** the degree of a node is the number of edges connected to it, or, in this context, the number of neighbors or connections of a process. In complex networks, the degree is used as one of the measures of centrality of a node, and the degree distribution is largely used to characterize a network. In this work, the average degree will be used to observe how the degree evolves along with the scalability of a parallel program.
- **Average shortest path length:** this metric measures the typical separation between two nodes in a graph and it is normally used to evaluate the transport of communication in a network. In a parallel program, this transport occurs in a regular way defined by the parallel algorithm. In the present work this metric will be used to examine how is the behaviour of the parallel program related to the distance among processes as the program scales.
- **Average clustering coefficient:** this metric is largely used, along with the degree distribution, to characterize real complex networks, like social networks, for example. It measures where two individuals with a common friend are likely to know each other [4]. In parallel programs this seems out of context, but the idea here is that it could help to understand the parallel program network if we can compare it to real networks.
- **Graph density:** this metric expresses the actual number of edges versus the number of edges that would be present if the graph was complete. In social networks, the density decreases as the network grows and our hypothesis is this is also desirable for scalable parallel programs.

3.1 Graph Visualization and Analysis

For graph visualization and analysis we chose Gephi [3], an open source network exploration and manipulation software. Developed modules can import, visualize, spatialize, filter, manipulate and export all types of networks.

Layout algorithms set the graph shape and it is the most essential operation. Graphs are usually laid out with force-based algorithms. They follow a simple principle: linked nodes attract each other and non-linked nodes are pushed apart. The Gephi tutorial, which could be found in [5], summarizes the layout choices, for example, OpenOrd (emphasis in divisions), ForceAtlas, Yifan Hu, Fruchterman-Reingold (emphasis in complementarities), and others.

The algorithms with emphasis in complementarities are used mostly to build readable graphs, which could be very convenient also to apply filters. For the representation of parallel programs our emphasis relies on the divisions or complementarities among communicating processes. The Yifan Hu layout [10], for example, has good results for large undirected graphs and could provide good visualizations for parallel programs as will be presented in this paper.

4 Case Study: Characterizing NAS Parallel Benchmarks

This section presents the methodology and the first results on characterizing parallel programs through graph visualization and complex networks metrics. The NAS parallel benchmarks [2] were chosen for its large use for testing the capabilities of parallel computers and parallelization tools. In order to present the case study, first we describe the methodology, followed by a brief description of NAS SP-MZ and BT-MZ benchmarks and finally the visualization and metrics for the characterization.

4.1 Methodology

In order to build a graph of communicating processes, it is necessary first to collect execution data: how many processes, how was the communication between them, and how was the distribution, the volume and the number of messages exchanged, for example. This task could be accomplished by using a trace file generator. Second, it is required to read the produced trace file or statistics file and generate another file with the textual representation of the graph in a specific graph format. This textual representation of the graph will be finally loaded to the graph visualization tool.

Thus, the methodology could be divided basically in two phases: graph building and graph visualization. In the graph building phase, first the communication data are collected from a trace file generator in the form of a communication adjacency matrix. Then, the graph is written in some specific textual format considering the processes as nodes and communication links as edges. The total number or volume of messages exchanged during the execution will be the weight of the edges.

In the graph visualization phase, the communication graph is loaded into a graph visualization tool that allows different layouts as well as the use of filters to visualize subgraphs. These subgraphs could be used to decrease the volume of visualized nodes if it becomes too large.

Different trace file generators could be used in order to extract information about the execution of a parallel program. Besides, there are some ways to instrument a program in order to generate data about specific events. Our approach in this work was to use EZTrace [18], a generic framework for performance analysis. In addition to provide an execution trace file, EZTrace also produces a statistic file and two communication adjacency matrix files. Since in this specific work we are interested in the communication size and in the number of messages exchanged, we used the EZTrace matrixes directly. For further works we intend to collect other events and even timestamps from the trace file.

Since EZTrace already provides the adjacency matrices, it is straightforward to build a textual representation for the graph. Several graph formats could be used for this task, like the ones supported by Gephi: GEXF, GDF, GML, GraphML, CSV, among others [5]. For simplicity, in this work we chose the GML format [9]. A script was written to read the adjacency matrices generated by EZTrace and write it in GML format. Nonetheless, in the future we intend to use GEXF format, which is a Gephi project. GEXF is XML based being more flexible and providing more features.

4.2 NAS Parallel Benchmark

The NAS Parallel Benchmarks (NPB) [2] are well-known problems for testing the capabilities of parallel computers and parallelization tools. They exhibit mostly fine-grain exploitable parallelism and are almost all iterative, requiring multiple data exchanges between processes within each iteration. The application benchmarks Lower-Upper Symmetric Gauss-Seidel (LU), Scalar Penta-diagonal (SP), and Block Tri-diagonal (BT) solve discretized versions of the unsteady, compressible Navier-Stokes equations in three spatial dimensions. Each operates on a structured discretization mesh that is a logical cube.

The NPB Multi-Zone versions of LU, BT, and SP benchmarks are LU-MZ, BT-MZ, and SP-MZ. In each, a logically rectangular discretization mesh is divided into a two-dimensional horizontal tiling of three-dimensional zones of approximately the same aggregate size as the original NPB. The major difference between the three multi-zone problems lies in the way the zones are created out of the single overall mesh [20].

The SP-MZ and BT-MZ versions are of specially interest in this work, since zones in each of the two horizontal dimensions grows as the problem size grows. The difference between the two implementations relies in the variation of zone sizes which only happens in BT-MZ implementation (in SP-MZ the zone size is fixed).

The implementation follows a stencil communication pattern with exchange of boundary values between zones taking place after each time step. Solution values at points one mesh spacing away from each vertical zone face are copied

to the coincident boundary points of the neighboring zone [2]. The problem is periodic in the two horizontal directions (x and y), so donor point values at the extreme sides of the mesh system are copied to boundary points at the opposite ends of the system. This property characterizes a toroidal system and was captured by Yifan Hu [10] graph layout algorithm as demonstrated later in this paper.

4.3 Characterization Results

This subsection presents some results in characterizing NAS parallel benchmarks SP-MZ and BT-MZ implementes with MPI [17]. The characterization is presented by means of visualization and network properties: average degree, average shortest path length, average cluster coefficient and graph density.

Characterizing SP-MZ. First, a characterization is presented for class C of NAS SP-MZ, which were executed with 32, 64, 128 and 256 processes in a single node. Figure 1 presents the visualization for the four cases with Yifan Hu graph layout provided by Gephi.

Despite the reduced visualization, it is possible to observe the regularity of the shape throughout different problem sizes. Also, it is likely to identify the cylindrical aspect that characterizes a regular toroidal stencil algorithm. As the problem grows, the cylinder volume also grows instead of the cylinder diameter. It is possible to infer by this visualization that this 3D characteristic helps the algorithm to scale since it preserves a more local communication pattern.

Table 1 presents the network metrics for SP-MZ. The average degree for 32 and 256 processes are absolute values; i.e., all nodes have exactly the same degree. It is interesting to notice that the average degree does not grow significantly with the size of the problem. This means that the number of each process connections stays practically the same. This seems to be a relevant feature of a scalable parallel program, since the communication overload per process is stable as the problem size grows.

The average shortest path length has a sublinear growth. According to [4], in regular hypercubic lattices in D dimensions, the mean number of vertices one has to pass by in order to reach an arbitrarily chosen node, grows with the lattice size as $N^{1/d}$. SP-MZ has a 3D toroidal structure and the measured average path length is just a little above this expectation. This metric seems to be a good reference for scalable parallel programs, since it denotes proximity among nodes.

The average cluster coefficient presented very small values, even 0 for two cases. These two cases are the most regular cases, where all nodes had exactly the same degree. In this particular case study, a low cluster coefficient seems to be an indication of regularity or balance in the communication pattern. However, more case studies would be necessary to observe the behavior of this metric in parallel programs.

The last metric analyzed was the density which decreases as the problem size grows. As discussed before, this metric expresses the actual number of edges

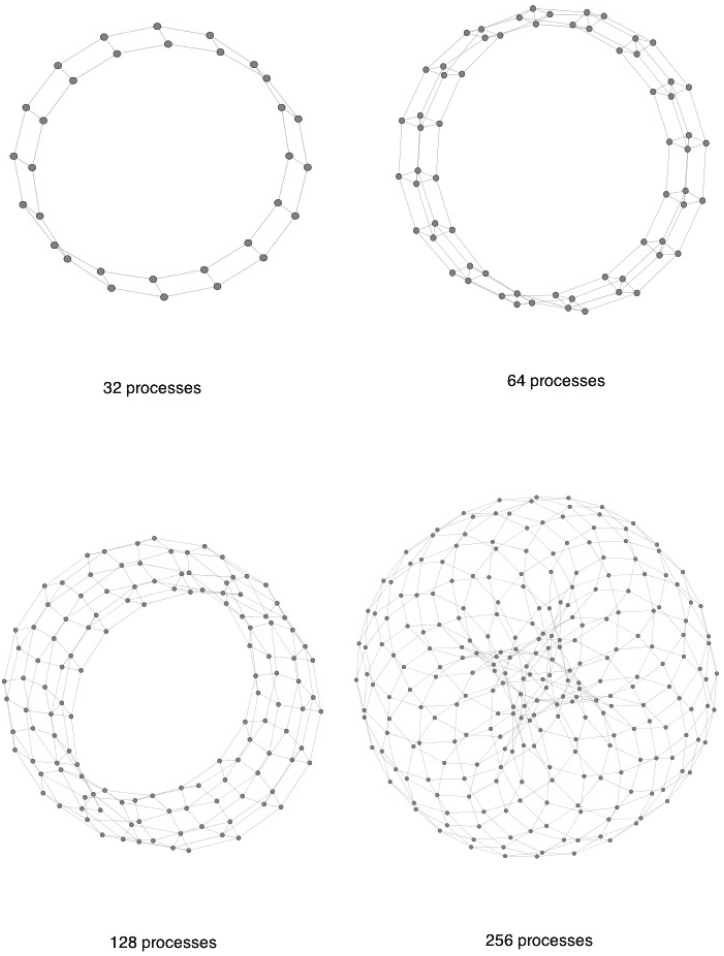

Fig. 1. Visualization of SP-MZ: (a) 32 processes, (b) 64 processes, (c) 128 processes and (d) 256 processes.

Table 1. Metrics for SP-MZ

Metric	32	64	128	256
Av. degree	*3.0*	4.5	4.25	*4.0*
Av. shortest path	4.64	4.95	5.85	8.03
Av. cluster coeff.	*0*	0.183	0.046	*0*
Density	0.097	0.071	0.033	0.016

versus the number of edges that would be present if the graph was complete. The decrease of density in this case means the relative number of connections

among processes decreases with the problem size. Since an excessive number of connections usually represents communication and synchronization overload in parallel programs, this characteristic seems to be desirable in parallel programs.

Characterizing BT-MZ. The BT-MZ benchmark has a more irregular behavior. The Fig. 2 shows the graph visualization for BT-MZ using the same layout algorithm as in SP-MZ (Yifan Hu graph layout provided by Gephi). It is possible to see that the shape is undefined in BT-MZ version, except for the last case.

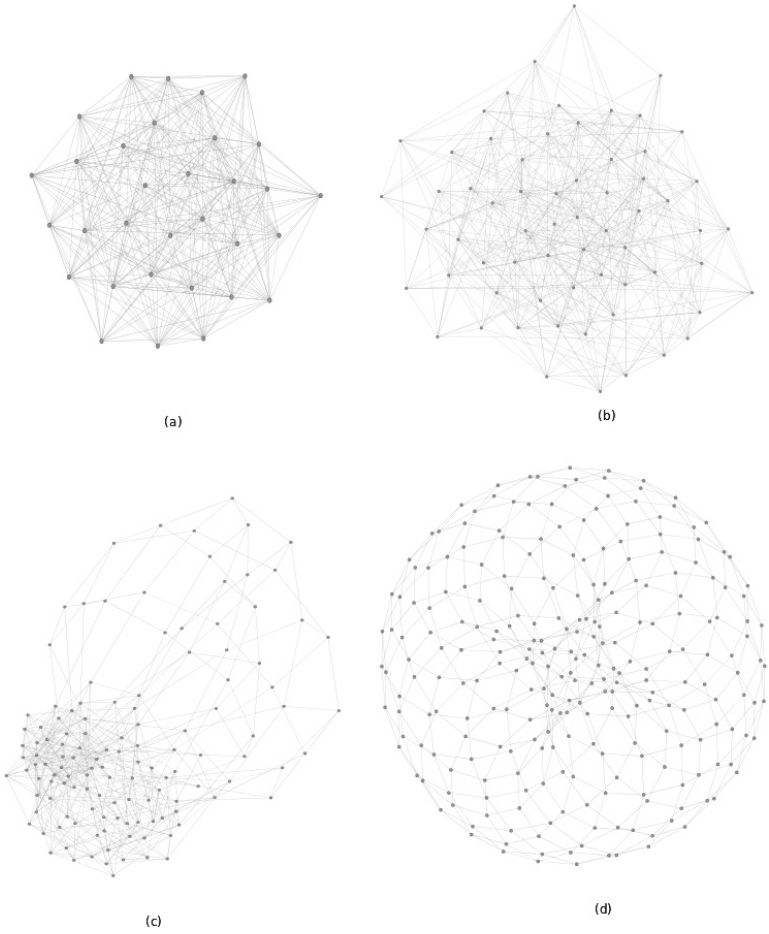

Fig. 2. Visualization of BT-MZ: (a) 32 processes, (b) 64 processes, (c) 128 processes and (d) 256 processes.

The Table 2 presents the metrics for the same problem sizes as for SP-MZ analysis. The average degree for 32 and 256 processes are absolute values; i.e., all nodes have the same presented degree.

Table 2. Metrics for BT-MZ

Metric	32	64	128	256
Av. degree	*20*	14.22	7.5	*4.0*
Av. shortest path	1.35	1.81	3.04	8.03
Av. cluster coeff.	0.64	0.21	0.06	*0*
Density	0.65	0.23	0.06	0.016

As mentioned before, BT-MZ implements variation of zone sizes which makes its structure more irregular and unbalanced than SP-MZ. This could be seen in its metrics, except for the last column, that is identical to SP-MZ for 256 processes. This means that for the last case of class C, the two versions have the same behavior related to the metrics analyzed and even considering the visualization.

The BT-MZ average degree is less stable than in SP-MZ, which is highly related to its greater irregularity. The average shortest path is inferior, except for 256 processes, while the average cluster coefficient and density are superior than in SP-MZ in most cases. As the metrics shows, it is possible to recognize characteristics that denote regularity or irregularity in the cases studied. Other studies are needed to advance in the use of these metrics for characterization and performance analysis.

5 Discussion and Future Work

The presented methodology is a preliminary work on using graph analysis tools to help in performance analysis of parallel programs. The goal is to offer a supplementary perspective of parallel applications, instead of replace any of the existent tools.

One of the main features of state-of-the-art analysis tools is the possibility to follow the events in a timeline, like for example, the traditional space-time diagram present in Vampir framework [13]. Our first approach presented here doesn't capture temporal aspects of the application. Instead, this first proposal approximates more with other traditional view, the Communication Matrix (also present in Vampir tool), but here only with final cumulative values. While the Communication Matrix is a flat 2D view, the produced graphs rather could provide topological 3D views, depending on the communication characteristics of the algorithm. Graph analysis tools, like Gephi, add different types of possible interactions for the analysis, like the generation of complex networks metrics presented here. Other suitable feature could be the use of filters, that we intend to explore in future works.

There are many possibilities yet to explore, one of them being the dynamic visualization. In order to accomplish this, we intend to capture timestamps for each communication event from the trace file and to add this information to the graph. This is possible, for example, by using the GEXF format (Graph

Exchange XML Format) [6]. This format is part of Gephi project and allows networks to be filtered with the timeline component.

6 Conclusion

This paper presents a first work on characterizing parallel applications through graph visualization and analysis. We presented a methodology where a graph of communicating processes is extracted from trace data and loaded to a graph visualization tool in order to perform the characterization. We used EZTrace to generate the trace data for NAS parallel benchmarks and Gephi graph visualization tool to analyze it.

The work was able to provide parallel program visualization and analysis for some network metrics. The characterization allowed to identify some appropriate features for parallel programs, like a stable average degree and low density. Also, it was possible to compare both implementations in terms of their structure. Even that these are preliminary results, it is possible to estimate the potential of the technique considering scalable parallel applications.

We intend to continue this work first by exploring larger versions of NAS parallel benchmarks. After that, we intend to explore different features like other layouts, filters and statistics as well as a dynamic visualization mechanism to add temporal visualization. Other possibilities like load and energy balance diagnostics will be also considered.

References

1. Bailey, D.H.: Nas parallel benchmarks. In: Padua, D.A. (ed.) Encyclopedia of Parallel Computing, pp. 1254–1259. Springer, New York (2011)
2. Baily, D., Barscz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Weeratunga, S.: The NAS parallel benchmarks. rnr-94-007.pdf (1994). http://www.nas.nasa.gov/assets/pdf/techreports/1994/
3. Bastian, M., Heymann, S., Jacomy, M.: Gephi: An open source software for exploring and manipulating networks. In: Proceedings of the International AAAI Conference on Weblogs and Social Media, San Jose (2009)
4. Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., Hwang, D.: Complex networks: structure and dynamics. Phys. Rep. **424**, 175–308 (2006)
5. The Gephi website (2015). http://www.gephi.org/
6. The GEXF website (2015). http://gexf.net/
7. Ghoshal, G.: Structural and dynamical properties of complex networks. Ph.D. thesis, University of Michigan (2009)
8. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco (2000)
9. Himsolt, M.: GML: A portable graph file format. Technical report, Universität Passau, 94030 Passau, Germany (1999). http://www.infosun.fim.uni-passau.de/Graphlet/GML/gml-tr.html
10. Hu, Y.F.: Efficient and high quality force-directed graph drawing. Mathematica J. **10**(1), 37–71 (2005)

11. Huck, K.A., Potter, K., Jacobsen, D.W., Childs, H., Malony, A.D.: Linking performance data into scientific visualization tools. In: Proceedings of the First Workshop on Visual Performance Analysis, VPA 2014, pp. 50–57. IEEE Press, Piscataway (2014)
12. Kim, J.S., Lilja, D.J.: Characterization of communication patterns in message-passing parallel scientific application programs. In: Panda, D.K., Stunkel, C.B. (eds.) CANPC 1998. LNCS, vol. 1362, pp. 202–216. Springer, Heidelberg (1998)
13. Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M.S., Nagel, W.E.: The vampir performance analysis tool-set. In: Resch, M., Keller, R., Himmler, V., Krammer, B., Schulz, A. (eds.) Tools for High Performance Computing, pp. 139–155. Springer, Heidelberg (2008)
14. Lee, I.: Characterizing communication patterns of NAS-MPI benchmark programs. In: Proceedings of the SOUTHEASTCON, pp. 158–163. IEEE, Atlanta (2009)
15. Mercier, G., Clet-Ortega, J.: Towards an efficient process placement policy for MPI applications in multicore environments. In: Ropo, M., Westerholm, J., Dongarra, J. (eds.) PVM/MPI. LNCS, vol. 5759, pp. 104–115. Springer, Heidelberg (2009)
16. Schmitt, F., Dietrich, R., Kuß, R., Doleschal, J., Knüpfer, A.: Visualization of performance data for MPI applications using circular hierarchies. In: Proceedings of the First Workshop on Visual Performance Analysis, VPA 2014, New Orleans, Louisiana, USA, 16–21 November 2014, pp. 1–8 (2014). http://dx.doi.org/10.1109/VPA.2014.5
17. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI-The Complete Reference, Volume 1: The MPI Core, 2nd edn. MIT Press, Cambridge (1998). (revised)
18. Trahay, F., Ru, F., Faverge, M., Ishikawa, Y., Namyst, R., Dongarra, J.: Eztrace: A generic framework for performance analysis. In: Proceedings of the CCGRID. IEEE, Newport Beach (2011)
19. Vetter, J.S., Mueller, F.: Communication characteristics of large-scale scientific applications for contemporary cluster architectures. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), Fort Lauderdale, Florida (2002)
20. van der Wijngaart, R., Jin, H.: NAS parallel benchmarks, multi-zone versions. rnas-03-010.pdf, July 2003. http://www.nas.nasa.gov/News/Techreports/2003/PDF/
21. Zamani, R., Afsahi, A.: Communication characteristics of message-passing scientific and engineering applications. In: Zheng, S.Q. (ed.) Proceedings of the IASTED PDCS, pp. 644–649. IASTED/ACTA Press, Phoenix (2005)