Edge Bundling for Visualizing Communication Behavior

Ronny Brendel, Michael Heyde, Holger Brunst, Tobias Hilbrich and Matthias Weber

Center for Information Services and High Performance Computing
Technische Universität Dresden, Germany
{ronny.brendel, holger.brunst, tobias.hilbrich, matthias.weber}@tu-dresden.de, heyde.michael@googlemail.com

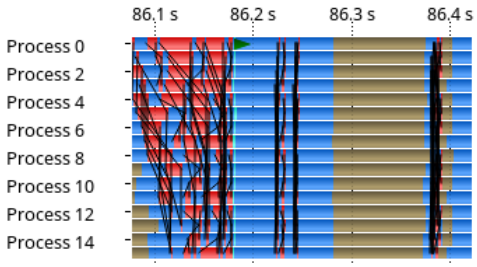
***Abstract*—To fully exploit the potential of today's computers, application developers need to design for concurrency. Along with parallel execution new performance problems emerge. Developers gain insight into application behavior by visualizing inter-process communication in timelines. They use this insight to eliminate performance bottlenecks. Timeline visualizations overlay function call structure with communication information and additional performance data. In many cases such visualizations suffer from information overload and visual clutter that complicate analysis.**

We address these problems by introducing techniques inspired by hierarchical edge bundling into time-based communication visualization. Our visualization combines individual messages into dominant communication paths and thereby highlights higher-level structures. Furthermore, we introduce scalable visualizations for communication profiles, which offer an alternative view on communication patterns. This work employs edge bundling at unprecedented scale to address emerging problems in timeline displays.

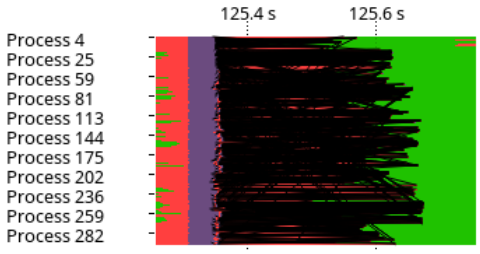
I. Introduction

Applications for High Performance Computing (HPC) systems employ parallelism to efficiently utilize the underlying hardware. Since current architectures feature distributed memory, message passing is a common approach to enable parallelism. The Message Passing Interface (MPI) [1] is the de-facto standard for this purpose and enables many applications that run on major HPC installations. Approaches to analyze and visualize the use of message passing aid application developers in their optimization and debugging workflows. Common tasks include detecting slow messages, load imbalances, inefficient communication patterns, and incorrect communication.

A common approach to support these tasks is to visualize the runtime behavior of the target application. For this purpose, timeline visualizations [2] display activities of individual processing elements. Tool suites such as Intel Trace Analyzer [3], Paraver [4] and Vampir [5] follow this idea and visualize processing elements as horizontal bars. Colors represent distinct application states and lines depict messages exchanged between processing elements. Fig. 1 presents two examples of timeline displays for an application using 16 processes (top) and an application with 1024 processes (bottom). This approach works well in the first example, Fig. 1a, and helps users identify individual communication phases. Users can then zoom into a specific communication phase to investigate individual messages. Fig. 1b depicts an inadequate visualization, where a large number of individual messages causes visual clutter. In this case, message visualization offers little insight and hides a large portion of the chart beneath. In situations as shown in Fig. 1b, visualizing all individual messages is not helpful.



(a) A *normal* example where message visualization—black lines—efficiently highlights interesting program behavior.



(b) An unfavorable example with a large number of messages that provide little helpful information, while hiding other parts of the chart.

Fig. 1. Timelines provide powerful visualization of the behavior and synchronization of parallel applications, however, increased event densities can be detrimental.

In this work, we present techniques to improve timeline visualizations, as well as other types of charts, to provide higher-level information. Our goal is to reduce message line clutter and emphasize communication patterns. We borrow ideas from edge bundling [6] which bends individual lines towards each other, and thus form bundles, in order to highlight dominant communication paths. A spline representation allows lines to deviate from their original straight representation. Our contributions include:

- A selection of real-world example applications with challenging communication patterns,

- The introduction of edge bundling into time-based communication visualization. For this we apply edge bundling to larger diagrams than any related work we are aware of,
- An alternative edge bundling approach that routes edges through a grid rather than a hierarchy, and
- An investigation of edge bundling for displays that summarize communication behavior of parallel applications.

The remainder of this work is organized as follows: Section II presents related work. We then discuss Vampir's current solution to combat message line occlusion and clutter in Section III. Section IV introduces the example applications we subsequently use. In Section V-A we explain in detail how to incorporate edge bundling into time-based communication visualizations. This is followed by an investigation into how edge bundling can improve communication profiles in Section V-B. We then summarize our experience with edge bundling and suggest future development directions.

II. Related Work

The presented work lives in the context of the performance visualization suite Vampir [5] and the corresponding monitor Score-P [7]. Vampir is specialized in visualizing highly parallel programs, and supports a multitude of paradigms, for example: MPI, POSIX Threads, OpenMP, CUDA, and SHMEM. By providing many different visualizations for function-, communication-, performance counter-, and I/O-related performance data, Vampir helps users find performance problems and gain understanding of their applications' performance characteristics.

A. Edge Bundling

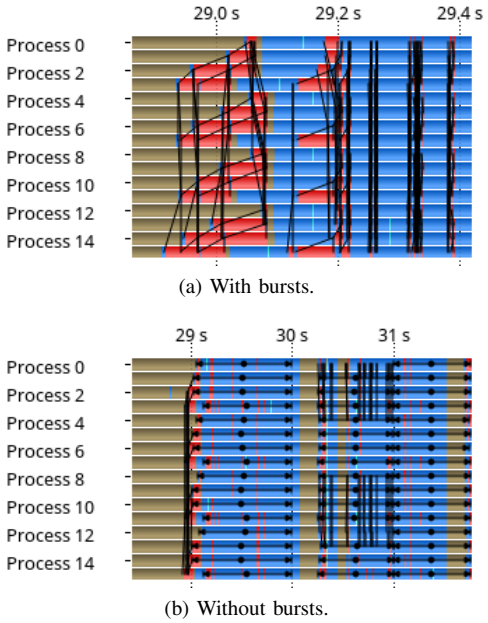
To reduce clutter and improve the visual appeal of trees with an additional adjacency relation defined on its nodes, Danny Holten introduced hierarchical edge bundling [6]. The technique achieves this goal by bending lines along the given hierarchy and therefore functions much like a cable harness. Our presented work is mainly inspired by this approach.

Later, Holten lifted the constraint of needing a hierarchy for edge bundling by introducing force-directed edge bundling [8]. In principle the technique assigns weight to edges and computes gravitation, which than leads to edges being drawn towards each other. Force-directed edge bundling yields a much more natural bundling than hierarchical bundling but is computationally more expensive.

Lambert et al. [9] present an alternative edge bundling approach, where the basis for control polygons is a quadtree. The quadtree resolution is finer in places where many lines are present. Routing is done via the shortest path. To reduce the computational demand, the authors introduce a hybrid quadtree/Voronoi approach. Furthermore they show benchmarks for an optimized CPU and an optimized GPGPU implementation.

B. Performance Visualization Tools & Techniques

A wide array of techniques exists for performance visualization and performance optimization [10]. One example for applied edge bundling, among other novel ideas, is the use of circular hierarchies for performance visualization [11]. The approach employs chord diagrams that incorporate information of the hardware topology and performance metrics to intuitively visualize communicating processes.



(a) With bursts.

(b) Without bursts.

Fig. 2. Message bursts reduce the number of displayed message lines in Vampir's Master Timeline.

Techniques to automatically detect computation phases [12] provide an intuitive improvement for timeline visualizations. This work defines computation phases to be time spans between communication operations. For these phases, the number of completed instructions is weighted against the elapsed time. Phases differ in how many instructions they complete in a given time window. This rate provides the input for a clustering that signifies distinct computation phases. The use of a visualization with per-cluster colors yields useful insights into the application's program structure. This approach works under the assumption that the application is homogeneous or of an SPMD structure.

Another take on time-based performance visualization is to order execution traces logically rather than strictly by time [13]. This allows a subsequent visualization that captures the developer's intended operation order better than usual timelines. The communication structure of the application determines the logical time. The logical time is then used to highlight communication patterns, e.g. for comparing and clustering similar processes. This visualization furthermore presents time-based metrics, such as lateness, in the logical program structure. Prior to this work, DeWiz [14] employed a similar approach.

To reduce visual obstruction in timelines due to communication lines, Oracle Solaris Studio [15] implements a simple slider for adjusting the number of visible messages.

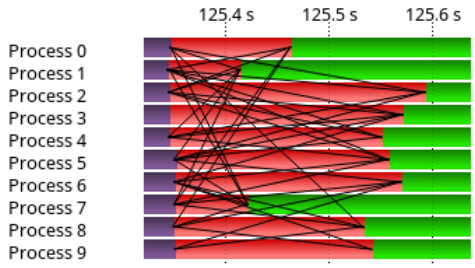


Fig. 3. Lines going every which way gives little insight into program structure.

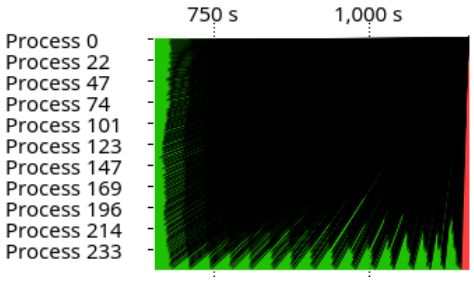


Fig. 4. In Tachyon, every processes repeatedly sends intermediate results to the first process. Here, the message visualization occludes almost all program state information beneath.

III. Motivation

The current solution in Vampir to avoid occlusion of program state information due to too many message lines is called *message burst*. Fig. 2 depicts an example application with and without message bursts. On a process, a burst consists of a start time, an end time, and a set of messages it represents. It signifies that a number of sending operations take place during the duration of the burst.

While this technique is effective in reducing the amount of message lines, it omits interesting information. Information such as duration and the target process of the replaced messages are not available. Message bursts are by design sender-oriented to enable a low-overhead mechanism that is scalable.

Fig. 3 shows a zoomed-in version of the timeline from Fig. 1b. The chart highlights that communication takes place and when it starts/ends, but it highlights no obvious communication pattern.

We suggest a time-based communication visualization that employs edge bundling to improve the short-comings discussed above. Our goal is to preserve interesting information while alleviating occlusion and reducing message line clutter.

IV. Test Cases

We use three HPC applications to highlight the behavior of our visualizations. The first application is from the WRF Model [16]. The application run utilizes 16 processes and sends about 160,000 point-to-point messages during a runtime of four minutes. The corresponding *execution trace* is available at https://www.vampir.eu/public/files/tracefiles/Large.zip. Fig. 1 and Fig. 2 depict Vampir's *Master Timeline* that shows zoomed-in portions of the overall trace.

The second application is FD4 [17], which introduces a dynamic load balancing scheme to couple the COSMO weather forecast model with the SPECS [18], [19] cloud microphysics scheme. In five minutes of runtime, FD4 exchanges about 2.2 million messages between 1,024 processes. This example presents a test case with an irregular communication pattern, resulting from the dynamic load balancing scheme, and an increased process count. Fig. 1b and Fig. 3 visualize portions of the run.

Finally, Tachyon [20] exhibits a single type of communication pattern only (Fig. 4). Processes only send messages to process 0, which in turn only receives messages. Tachyon is part of the SPEC MPI2007 benchmark suite [21] and implements ray tracing for image rendering. The decomposition of this application lets processes render parts of the image independently. The first process then gathers the individual results to combine them into the final image. Our example run utilizes 256 processes and exchanges about 8000 messages during a runtime of 20 minutes.

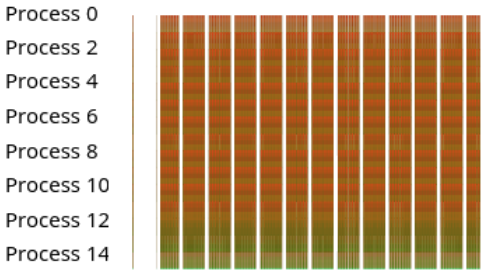
V. Edge Bundling

This section initially demonstrates how edge bundling can improve timeline communication visualization. We subsequently explore how the developed techniques can be used to aid visualizing communication profiles.

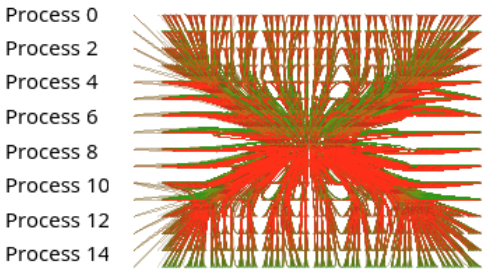
A. Time-based Visualization

We use timeline visualizations as a promising target to apply hierarchical edge bundling [6]. Fig. 5a depicts a timeline for the full WRF run without any reduction techniques. Fig. 5b and Fig. 5c show hierarchical edge bundles instead of straight message lines. The charts arrange the 16 processes vertically and let time run from left to right. We employ B-Splines of an adjustable degree—we use a degree of two throughout this paper—to represent the messages. To discern sender and receiver processes, we color splines with a color gradient from green (sender) to red (receiver). When splines overlap, we apply alpha blending to emphasize dominant communication paths.

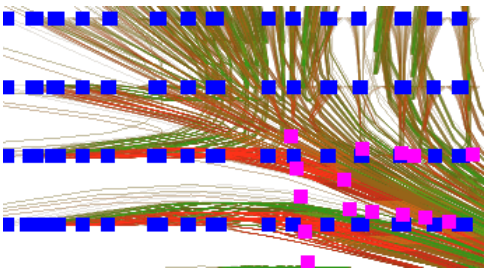
We apply our hierarchical edge bundling as follows: in order to create a hierarchy for the routing of the splines, we take all send- and receive-events on their respective process and cluster them using the mean shift [22] algorithm. The resulting cluster centers constitute the lowest hierarchy level. We recursively cluster the centers of each hierarchy layer to generate a full hierarchy with a single root. Each hierarchy node is connected to its cluster center, thus ultimately creating a tree. In our prototype implementation we use an adjustable number of hierarchy levels, which is set to three in the examples. The mean shift algorithm then determines the number of clusters per level. Each send-event creates a spline that starts at the event, that we route through the hierarchy to the target receive-event. To illustrate this scheme, Fig. 5c presents the cluster centers of our hierarchy for the upper-left portion of the timeline in Fig. 5b. The lowest hierarchy level uses blue boxes



(a) Without edge bundling, and without message bursts.

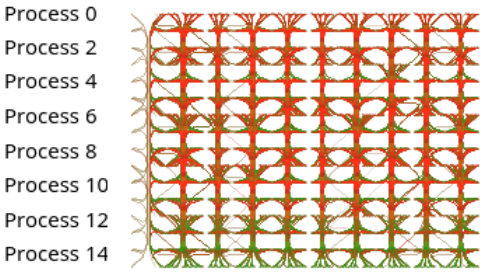


(b) With hierarchical edge bundling.

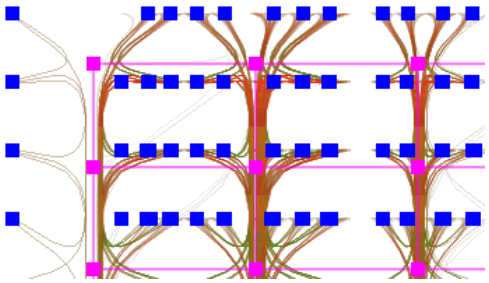


(c) Zoomed-into version of Fig. 5b with visible hierarchy nodes.

Fig. 5. The Master Timeline visualization for a WRF application run with 16 processes.



(a) Overall communication trace.



(b) Close-up with visible grid lines.

Fig. 6. Timeline with grid-based edge bundling for the WRF trace.

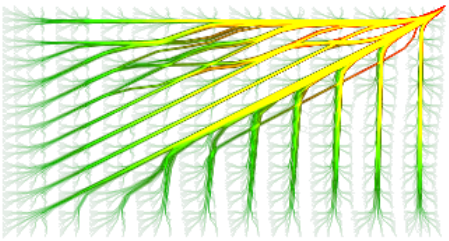


Fig. 7. Timeline with grid-based edge bundling and additive color mixing for the Tachyon run. Yellow bundles highlight hot paths.

for the cluster centers and the next level uses magenta for its centers.

The edge-bundled timeline saves screen space in comparison to directly visualizing messages as lines (Fig. 5a). However, this visualization has shortcomings. Since messages bend along tree edges, many messages have to bend along the root node, i.e. the center of the timeline. As Fig. 5a illustrates, few messages actually run through the center of the visualization. Thus, our visualization with hierarchical edge bundles can be misleading. In fact, in Fig. 5c, many messages are bent towards the right only to be bent to the left again, since our hierarchy dictates it. We therefore conclude that hierarchical edge bundling produces unintuitive images and is thus not ideal for timeline communication visualization.

As a result, routing large numbers of edges through the root of the hierarchy is not desirable. Thus, we target a routing directly through the lower hierarchy levels instead. For this purpose we use a grid that creates the control polygons, instead of a hierarchy. Fig. 6 shows the impact of this change for WRF. Our lowest-level hierarchy layer still consists of the clustered communication endpoints (blue boxes in Fig. 6b) as is the case for hierarchical edge bundling. The grid then forms the next level (magenta). We route each edge through the first level to the grid. Within the grid we route the edge along the shortest path that connects to the lower-level endpoint of the communication target.

To improve the edge visualization within our grid we extend our basic approach with the use of diagonals through the grid. Fig. 7 combines this extension and applies an additive color mixing instead of the previously used alpha blending. This form of composition highlights hot communication paths in bright yellow.

In comparison to our initial hierarchical approach, grid-based edge bundling provides a more intuitive visualization. Edges do not bend as far horizontally, the line arrangement is more tidy, and especially for the Tachyon run the visualization highlights the overall communication pattern very well. Due to the limitation that our grid is fixed in position and cell size, Fig. 6b shows that some lines are still bent more than necessary (leftmost vertical messages). We expect that variable position and cell sizes coupled with a quadtree refinement, as introduced by Lambert et al. [9], would further improve the diagram quality.

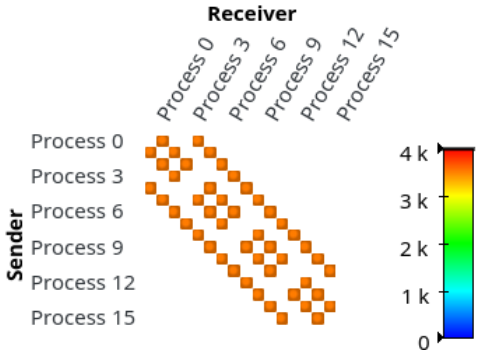
During our experiments it turned out that parameter tuning for the diagrams proves challenging. Picking a good opacity, bending degree, line width, color composition mode, hierarchy depth and number of grid dimensions heavily depends on the execution trace to be visualized. The choice of these parameters strongly influences the visual outcome and therefore makes or breaks the visualization. A production implementation needs to automatically adjust these parameters.

Concluding our investigation of how edge bundling can improve time-based communication visualization, we state: First, providing high diagram quality is hard, and requires manual parameter tuning so far. Second, edge bundling was originally conceived for creating more accessible diagrams of static data, like software module inter-dependencies [6]. It is intended for small-scale, non-interactive visualizations and most related work uses the techniques in this fashion. On the other hand, our goal was to study highly interactive visualizations for millions of communication events. Such an event count stresses current approaches for edge bundling. With our sequential prototype implementation we can compute and render the previous visualizations for WRF within about one minute. The same images for the FD4 examples, which are still multiple times smaller than other execution traces that Vampir can visualize, takes well over twenty minutes. While optimization for efficient and parallel processing are not a target for our current prototype, the total number of communication events in highly parallel applications will still render edge bundling too computationally demanding. We are not aware of existing approaches that bundle as many edges as we do for our examples. Lambert et al. [9] processed up to 16,000 edges in roughly one second with a highly optimized GPU-based solution and five or more seconds on a CPU.

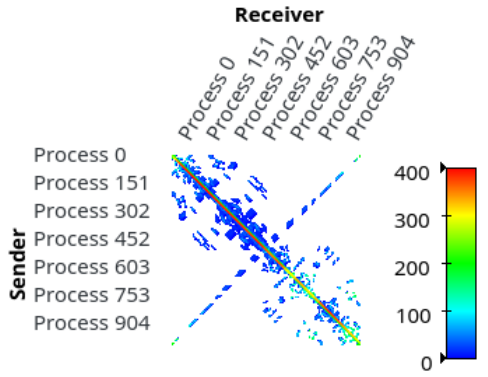
B. Summary Visualizations

While our edge bundling approach works well for small application runs, and while our grid-based bundling provides intuitive visualizations, we want to provide an approach that can work well with larger application runs, too. Thus, instead of visualizing all individual messages, we alternatively display one summarized value per pair of communication partners. For this purpose, Vampir offers the *Communication Matrix View*. Each cell of the matrix encodes a value, in color, with the sending process on the left and the receiving process on the top. Values such as average bandwidth, accumulated number of bytes sent, or the total number of messages sent then provide the values that we visualize. Fig. 8 shows such a diagram for WRF and FD4. The depicted value is the total number of messages sent. For Tachyon, since every process communicates with process 0 only, there is a vertical line of cells in the left-most column.

An alternative way to encode this information is to use chord diagrams together with hierarchical edge bundling, where each process is a node on a circle and lines between two nodes encode a quantity for this pair of processes. For simplicity, we do not use such an encoding and instead use our previous green-to-red gradient that distinguishes origin processes from



(a) Every process for the WRF run sends ~4,000 messages to two to four other processes.

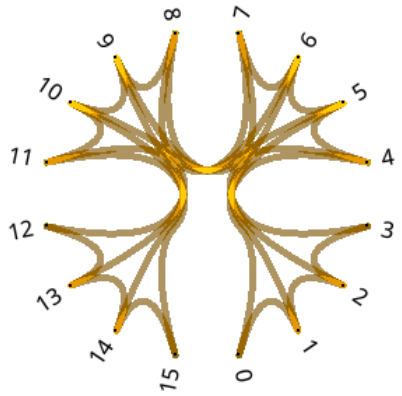


(b) FD4 shows a more diverse communication structure. Neighboring processes exchange most messages. Some communication takes place between farther away processes with respect to their process identifiers.

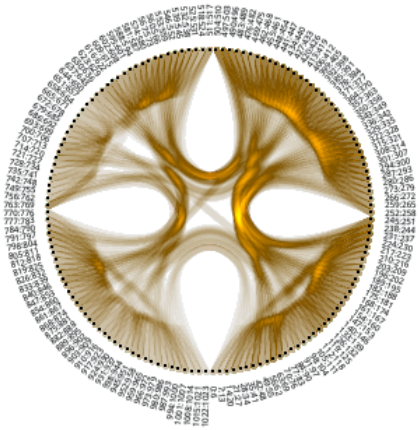
Fig. 8. The Vampir Communication Matrix displays communication statistics for a program run. It encodes one value in color for each pair of processes.

destination processes. Fig. 9 shows the result of our approach for the three example applications. For demonstration purposes, we use additive color mixing for these diagrams. If pairs of processes exchange messages in both directions, additive color mixing leads to homogeneously brown lines as depicted in Fig. 9a and Fig. 9b. The same effect can be observed in the time-based case too, if there are overlapping messages in opposing directions. As in the underlying communication matrices, the diagrams provide a visual impression about the complexity of the underlying communication pattern. Both the increased number of communicating processes in FD4, as well as the "towards-master" pattern of Tachyon are immediately visible. The diagram for FD4 (Fig. 9b) exhibits brighter edges on its right side. This highlights that half the processes have more communication partners than the other half. This is also visible in Fig. 8b with an increased number of cells in the upper left of the diagram. For the Tachyon application, each process sends messages to process 0, which the "hot" path in Fig. 9c highlights.

In contrast to our timeline-based visualizations, the number of messages to be drawn in our chord diagrams is limited by the squared number of processes (at most one edge for each pair of processes). Thus in practice, the number of lines for these diagrams is smaller than in the timeline visualizations.



(a) WRF: Each pair of processes exchanges messages in both directions. Therefore, additive color mixing leads to a homogenous brown color.



(b) FD4 shows the same brown color as WRF. The right side is brighter, which is a clear hint that there is more communication in this range of processes.

(c) Tachyon: If there are few overlapping edges in opposing directions, additive color mixing highlights hot paths.

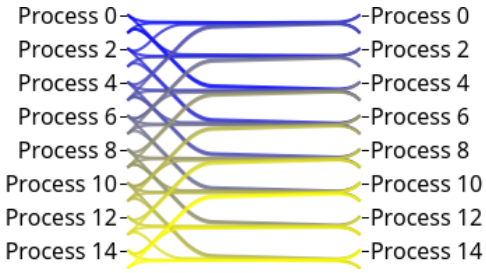
Fig. 9. Chord diagrams for the three example application runs with a green-to-red (source-to-target) gradient and additive color mixing.

In our examples, WRF has 48 pairs of communicating process and FD4 is down to ~60,000 edges compared to the 2.2 million edges in the timeline.

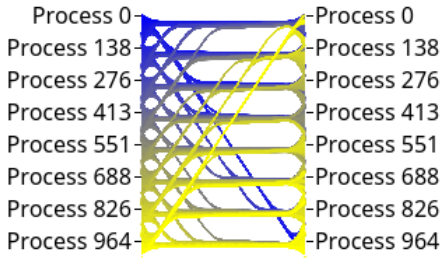
To improve our ability to highlight communication patterns, we introduce an alternative process arrangement. In a chord diagram, if two spatially close processes communicate, the diagram arrangement provides little opportunity to bundle this edge with other incoming edges. We propose, the *sender/receiver diagram* (Fig. 10), which arranges processes vertically and draws lines between sending processes on the left and the corresponding receiving processes on the right. This approach "stretches" the edges even for processes that would otherwise be spatially close in a chord diagram, thus providing better opportunity to bundle dominant communication paths. As for our extended timeline visualization, we use a grid to create control polygons. Edge routes are determined by an YX-routing preferring diagonals. Routing in this fashion has two key advantages over using the shortest path. First, it is deterministic, which the shortest path is not since there can be multiple paths of the same minimum length. Second, the routing highlights process pairs that are spatially distant as far running diagonals. Otherwise, edges transition into horizontal lines at an earlier point, e.g. in the first half of the diagram. We use a grid size of eight by eight, so it divides the process counts in our examples, i.e., 16, 256, and 1,024. Since we represent each process with two nodes in this diagram type, we do not need to use a color gradient to distinguish senders from receivers. Instead, we employ the color to identify the sending process.

For the WRF example, Fig. 10a highlights that edges do not stretch very far diagonally before they continue horizontally. This, and the fact that the color gradient extends not far vertically, provides a good visual impression of the property that processes do not communicate across large distances, but rather in their neighborhood. The diagram for the FD4 case (Fig. 10b) highlights that most processes send messages to neighboring processes. Additionally, a few processes communicate across larger distances. Fig. 10b represents the Tachyon communication pattern very clearly. As a fourth example, we add a trace of the LULESH [23] pseudo-application, which we ran on eight processes. This application run lets all processes communicate with all other processes. Again, the sender/receiver diagram in Fig. 10d highlights this pattern very well.

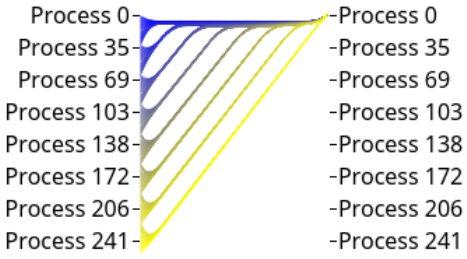
The challenge of parameter choice, as for our timeline visualizations, still applies. While the total number of messages to visualize is lower than for the timeline visualizations. Increased application scale upwards of 1,000 processes is challenging. For example, our sequential prototype requires about 30 seconds to generate the diagram inFig. 10b. At the same time, communication matrix views also suffer from a time complexity that increases with scale. Given our experiments, we perceive the sender/receiver diagram to be a simple, easily accessible, and powerful visualization for an application's underlying communication pattern. We see an improvement over chord diagrams.



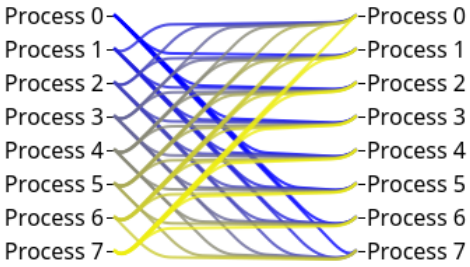
(a) WRF: Processes only exchange messages with their neighbors.



(b) FD4: Neighborhood communication is predominant. Some global communication takes place in addition.



(c) Tachyon: All-to-one.



(d) LULESH: All-to-all.

Fig. 10. The sender/receiver diagram for four example application runs. By coloring processes distinctly, users get a good sense of where messages are sent to.

To further improve our sender/receiver diagrams, we suggest to incorporate topology information. For an application developer, arranging processes based on the domain decomposition in use is helpful. Often, tools can extract this information from MPI communicators. Additionally, to understand hardware effects, the use of the hardware topology is an alternative source for improved diagram arrangements. Thus, we could route edges along the topology information, to better reflect the underlying system.

VI. Conclusion

We present an investigation of using edge bundling for both time-based and summary-based visualizations of parallel communication. Our experience shows that purely hierarchical edge bundling often produces unintuitive diagrams, since too many edges pass through the root of the hierarchy. To alleviate this problem, we introduce a grid-based approach that yields good visual results.

For timeline visualizations with edge bundling we see two major challenges: First, the total number of messages limits applicability to large application runs. Second, a broader suite of test cases and improved diagram quality must ensure that the charts provide reliable and intuitive visualizations for the majority of applications. Another area of focus lies in optimizing the performance of our techniques. Parallel algorithms could drastically reduce the time needed to generate charts. We propose the sender/receiver diagram to reduce runtime complexity, while improving the ability to highlight communication patterns over existing summary-based communication visualizations.

In short, we perceive that: First, edge bundling is hard to get right. The resulting diagrams need to be intuitive and helpful. Second, using edge bundling for interactive visualizations of large edge counts proves challenging. Edge bundling is usually used with at most 16,000 edges. In this work we present diagrams with up to 160,000.

VII. Future Work

In the short term, we aim to first improve the visual result quality of the sender/receiver diagram. Another opportunity for improvement is to encode values into edges, e.g., by varying their thickness. Additionally, we want to explore alternative ways to obtain control polygons, e.g., from the hardware topology.

Edge bundling for time-based diagrams is still a promising visualization type. Steps to improve our current approach include performance enhancements with techniques such as message filtering or by combining batches of related messages. Secondly, the grid technique needs to take the underlying communication structure into account instead of having a fixed parameter set.

We will also explore edge bundling techniques that do not require control polygons, like for example force-directed edge bundling [8].

References

[1] "Message Passing Interface (MPI) forum," http://mpi-forum.org, Apr. 2016.

[2] L. Lamport, "Time Clocks, and the Ordering of Events in a Distributed System," *Commununications of the ACM*, vol. 21, pp. 558–565, 1978.

[3] "Intel Trace Analyzer and Collector," https://software.intel.com/intel-trace-analyzer, Jul. 2016.

[4] V. Pillet, J. Labarta, T. Cortes, and S. Girona, "Paraver: A tool to visualize and analyze parallel code," in *Proceedings of WoTUG-18: Transputer and occam Developments*, vol. 44, 1995, pp. 17–31.

[5] H. Brunst and M. Weber, "Custom hot spot analysis of HPC software with the Vampir performance tool suite," in *Tools for High Performance Computing 2012*. Springer, 2013, pp. 95–114.

[6] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 741–748, 2006.

[7] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony *et al.*, "Score-P: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir," in *Tools for High Performance Computing 2011*. Springer, 2012, pp. 79–91.

[8] D. Holten and J. J. Van Wijk, "Force-directed edge bundling for graph visualization," in *Computer Graphics Forum*, vol. 28, no. 3. Wiley Online Library, 2009, pp. 983–990.

[9] A. Lambert, R. Bourqui, and D. Auber, "Winding roads: Routing edges into bundles," in *Computer Graphics Forum*, vol. 29, no. 3. Wiley Online Library, 2010, pp. 853–862.

[10] K. E. Isaacs, A. Giménez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer, "State of the art of performance visualization," *EuroVis 2014*, 2014.

[11] F. Schmitt, R. Dietrich, R. Kuß, J. Doleschal, and A. Knüpfer, "Visualization of performance data for MPI applications using circular hierarchies," in *Proceedings of the First Workshop on Visual Performance Analysis*. IEEE Press, 2014, pp. 1–8.

[12] J. Gonzalez, J. Gimenez, and J. Labarta, "Automatic detection of parallel applications computation phases," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–11.

[13] K. Isaacs, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer, "Ordering traces logically to identify lateness in message passing programs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 829–840, 2015.

[14] C. Schaubschlaeger, D. Kranzlmüller, and J. Volkert, "Event-based program analysis with DeWiz," *arXiv preprint cs/0310007*, 2003.

[15] "Oracle solaris studio 12.3: Performance analyzer MPI tutorial – viewing message details," https://docs.oracle.com/cd/E24457_01/html/E22002/ggxsx.html, Apr. 2016.

[16] J. Michalakes, S. Chen, J. Dudhia, L. Hart, J. Klemp, J. Middlecoff, and W. Skamarock, "Development of a next generation regional weather research and forecast model," in *Developments in Teracomputing: Proceedings of the Ninth ECMWF Workshop on the use of high performance computing in meteorology*, vol. 1. World Scientific, 2001, pp. 269–276.

[17] M. Lieber, V. Grützun, R. Wolke, M. S. Müller, and W. E. Nagel, "Highly scalable dynamic load balancing in the atmospheric modeling system COSMO-SPECS+FD4," in *International Workshop on Applied Parallel Computing*. Springer, 2010, pp. 131–141.

[18] B. Rockel, A. Will, and A. Hense, "The regional climate model COSMO-CLM (CCLM)," *Meteorologische Zeitschrift*, vol. 17, no. 4, pp. 347–348, 2008.

[19] V. Grützun, O. Knoth, and M. Simmel, "Simulation of the influence of aerosol particle characteristics on clouds and precipitation with LM-SPECS: Model description and first results," *Atmospheric Research*, vol. 90, no. 2, pp. 233–242, 2008.

[20] "122.tachyon SPEC MPI2007 benchmark description," http://www.spec.org/mpi2007/docs/122.tachyon.html, Jun. 2016.

[21] M. S. Müller, M. van Waveren, R. Lieberman, B. Whitney, H. Saito, K. Kumaran, J. Baron, W. C. Brantley, C. Parrott, T. Elken *et al.*, "SPEC MPI2007–an application benchmark suite for parallel systems using MPI," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 2, pp. 191–205, 2010.

[22] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Transactions on information theory*, vol. 21, no. 1, pp. 32–40, 1975.

[23] I. Karlin, A. Bhatele, J. Keasler, B. L. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang *et al.*, "Exploring traditional and emerging parallel programming models using a proxy application," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 919–932.