The Visual Display of Parallel Performance Data

Michael T. Heath
University of Illinois, Urbana-Champaign

Allen D. Malony
University of Oregon

Diane T. Rover
Michigan State University

A high-level abstract model lets visualization designers create displays in an integrated environment. The model directly links these displays to parallel performance models.

The primary motivation for using parallel computer systems is their high performance potential, but that potential is notoriously difficult to realize, and users often must analyze and tune parallel program performance. Parallel systems can be instrumented to provide ample feedback on program behavior, but because of the volume and complexity of the resulting performance data, interpreting these systems can be extremely difficult. Hence, performance tools are needed to help bridge the gap between raw performance data and significant performance improvements.

Data visualization has proved effective in deciphering many types of scientific and engineering data and facilitating human comprehension of large, complex data sets. Some of the most successful parallel performance tools are based on visualization techniques. The visual representation of data for a physical system is usually based on physical concepts and models that are intuitively meaningful to the user. The interpretation of performance data, on the other hand, involves a seemingly artificial, abstract model of parallel computation that may have little or no direct meaning to the user and may be difficult to relate to application-level concepts represented by the user's program.

Despite these difficulties, several performance visualization tools have demonstrated that helpful insights into parallel performance can be gained through graphical displays. However, much of this work has been experimental, specialized, and ad hoc. Evolving performance visualization into an integral, productive tool for evaluating parallel performance requires a more systematic, formal methodology that relates behavior abstractions to visual representations in a more structured way. In this article, we propose a high-level abstract model for the performance visualization process, explain its relationship to the most important concepts and principles of effective visualization practice, and illustrate the relationship between these concepts and our abstract model through specific case studies. We also discuss the relationship of performance visualization to general scientific visualization.

PARALLEL PERFORMANCE VISUALIZATION MODEL

The model we propose for parallel performance visualization is based on two important principles:

- Visual displays of performance information should be linked directly to parallel performance models.
- Performance visualizations should be designed and applied in an integrated environment for parallel performance evaluation.

To clarify the abstract model, we first give a concrete example of the process and then generalize it. Good parallel performance requires, among other things, that the computational work be spread evenly across the processors, that each processor do its share concurrently, and that addi-

0018-9162/95/$4.00 © 1995 IEEE

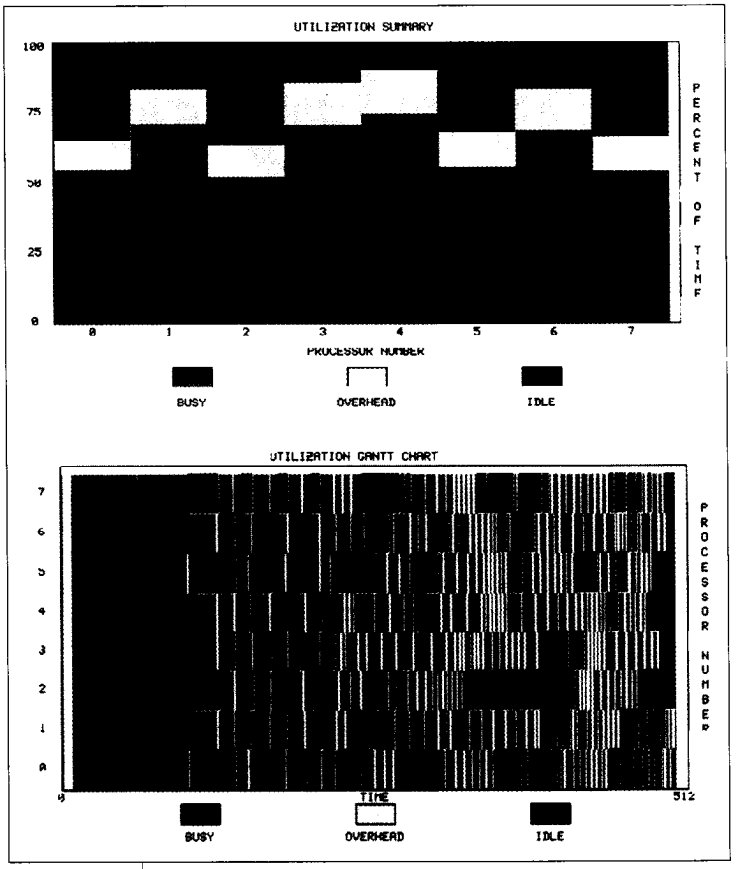


Figure 1. An example from ParaGraph illustrating the performance visualization process model: (top) utilization summary histogram; (bottom) utilization Gantt chart.

tional work beyond that required by a serial algorithm be minimized. Thus, the corresponding processor-oriented performance analysis abstractions—load balance, concurrency, and overhead, respectively—are important for optimizing performance. For quantification, these abstractions must be mapped to appropriate parallel performance data that can be monitored, such as processor busy, idle, and communication times. For visualization, these performance analysis abstractions must also be mapped to appropriate visual abstractions, such as bar charts and histograms. The final step, possibly after intermediate processing, involves mapping the performance data to specific instances of the visual abstractions to obtain a performance display, as exemplified by Figure 1 (produced by ParaGraph[1]).

At the top of this figure, the data are integrated over time, giving a quick and effective visual impression of load balance and overhead but no insight into concurrency. Although this display can identify poor load balance or excessive overhead, it does not pinpoint the specific time of its occurrence. This deficiency is remedied by the display shown at the bottom of Figure 1, where the same information is given as a function of time, so that specific periods of activity and idleness on specific processors can be identified and correlated across processors. However, this more detailed view is less effective in providing an overall impression of relevant performance abstractions.

The high-level parallel performance model that we propose (depicted graphically in Figure 2) emphasizes the binding of performance analysis abstractions to performance visual abstractions. The terms in this figure are defined as follows:

- *Performance analysis abstraction*—a specification of the performance characteristics to be observed from the data, the performance analysis to be performed, and the semantic attributes of the performance results.
- *Performance view*—a representation of a performance analysis abstraction such that its attributes can be mapped to a performance display.
- *Performance visual abstraction*—a specification of the desired visual form of the abstracted performance data, unconstrained by the limitations of any particular graphics environment.
- *Performance display*—a representation of a performance visual abstraction in a form that identifies the visual properties to which the attributes of a performance view are mapped.
- *Performance visualization abstraction*—the mapping of a performance view to a performance display.

A key point here is that the performance visual design can and should incorporate knowledge of the performance analysis abstraction very early on (as indicated by the dashed horizontal arrow in Figure 2), providing the basis for performance interpretation in the final visualization. The binding between performance analysis and visual abstractions is a mapping from performance view outputs to performance display inputs. This abstraction embodies the integral relationships between performance data and visual display that reveal the visualization's performance meaning.

So that it will be useful and its effectiveness can be evaluated, a performance visualization abstraction must be instantiated in a performance visualizer tool. This tool implements performance views, displays, and the mappings between them, using environment-specific graphics technology based on underlying graphics libraries, toolkits, and other resources.

VISUALIZATION CONCEPTS AND PRINCIPLES

The concepts and principles underlying good data visualization are becoming reasonably well understood.[2-4] Some of these concepts have proved useful in the design of effective performance views and displays. Still others have arisen from the unique challenges of interpreting performance data.

Context

To present performance information clearly, we must establish some context to which users can relate that information:

- *Perspective*—the point of view from which information is presented. Typical perspectives for performance information include the hardware, the operating system, and the application program. A

given perspective may emphasize states of processes or processors, or interactions among them, and the information may or may not be amalgamated over space or time. For example, Figure 1 shows two views depicting processor states.

- *Semantic context*—the relationship between performance information and the constructs and abstractions (such as data structures and control structures) in the application program. For example, Figure 3, produced by Popeye,[5] shows performance information (the locality of memory references) for a specific data structure (a two-dimensional array) in an application program. Another example is selecting a graphical image from an element and highlighting the corresponding portion of the application program. For instance, in the Automated Instrumentation and Monitoring System (AIMS),[6] selecting a communication line between processors highlights the corresponding send and receive statements in the user program.
- *Subview mapping*—a mapping between a subset of a graphical view (for instance, a rectangular subregion) and the corresponding subset of the data being rendered.[7] This mapping implies that the data can be reconstructed from the image, which would not be the case if the data were reduced so that details were lost. For example, each horizontal bar in Figure 1 (bottom) depicts the detailed time history of the corresponding processor, whereas in Figure 1 (top), such detail is absent because the data have been averaged over time.

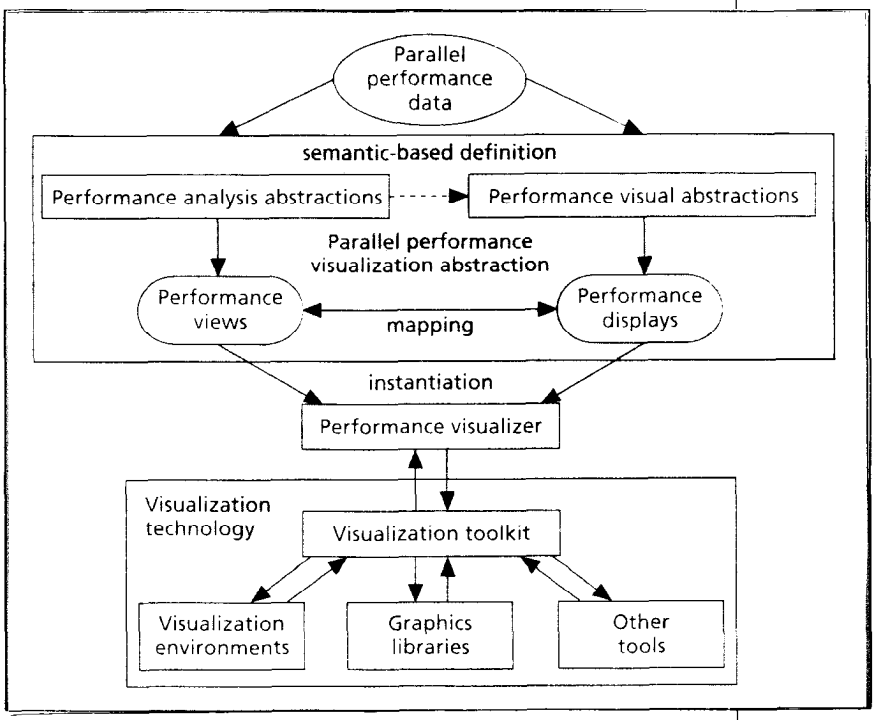


Figure 2. The parallel performance visualization model emphasizes the binding of performance analysis abstractions to performance visual abstractions.

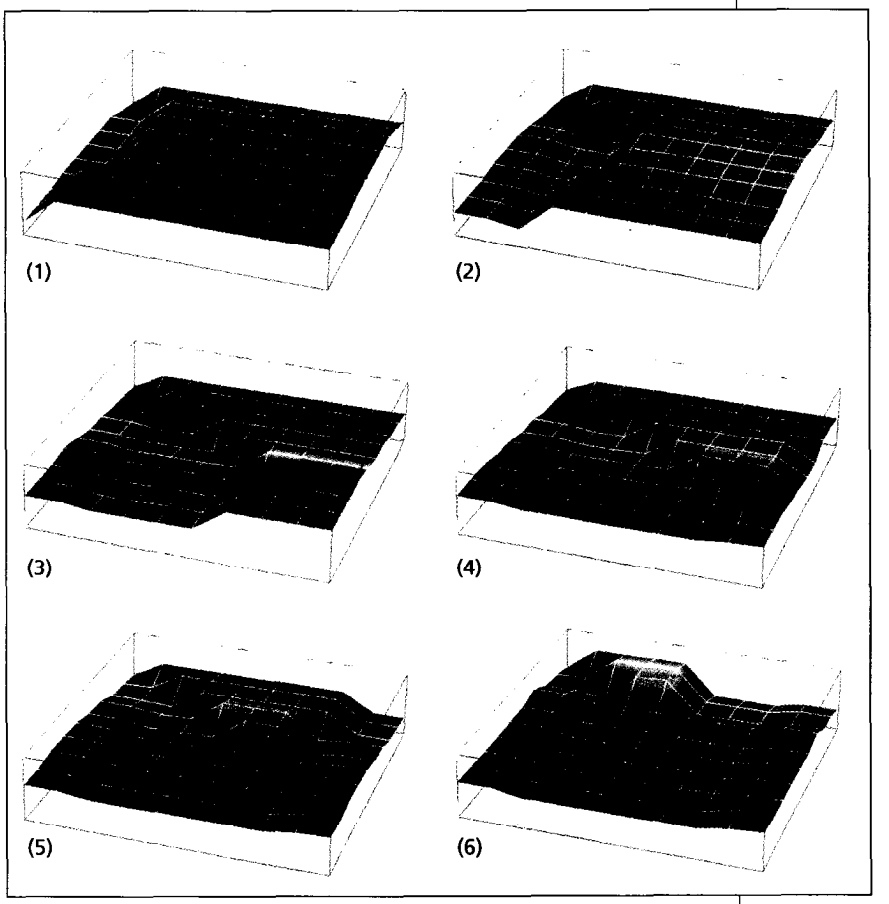
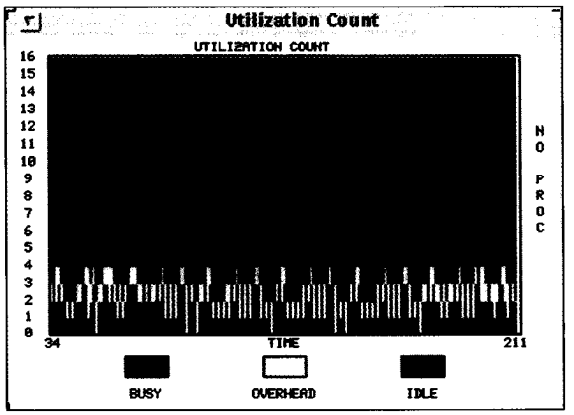


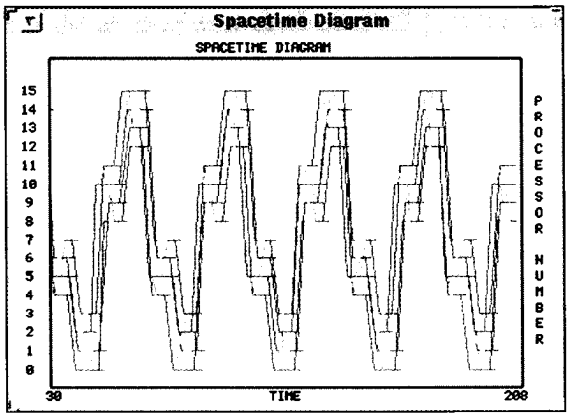
Figure 3. Viewing data accesses: time series of data-array-access surface plots (generated using IBM Data Explorer).

Scaling

For scientific visualization, scaling graphical views as data sets become very large is a major challenge. This is especially true of performance visualization as the number of processors or duration of execution substantially increases. Several techniques have been used to handle this scaling problem:

- *Multidimensional and multivariate representation*—a representation of data with many attributes per data point. Like most scientific data, performance data are typically multidimensional, with both space-like (for example, processors or memory) and





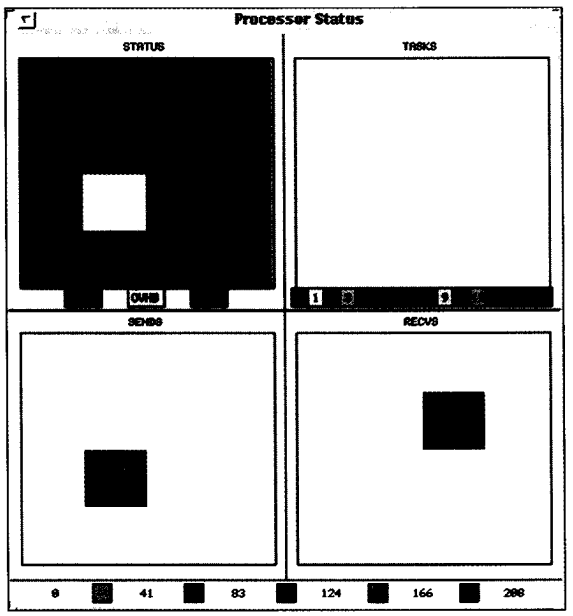


Figure 4. Viewing concurrency patterns: (top) utilization count display; (middle) space-time diagram from ParaGraph; (bottom) processor status display, where top left grid represents processor state, bottom left represents volume of messages currently being sent by each processor, and bottom right represents volume of messages currently being received by each processor; the top right grid is not used in this example.

time-like dimensions and other parameters (such as problem size) that may vary as well. Such a multivariate representation is conceptually compact, but the technical challenge for visualization is to represent as many dimensions as possible on a flat video screen. Figure 1 (bottom) uses two screen dimensions plus color to depict three data dimensions (time, processor, and state), whereas Figure 3 uses an explicit three-dimensional rendering, with color reinforcing the vertical dimension.

- *Macroscopic and microscopic views*—the level of detail represented by a given view. A macroscopic view conveys the big picture, while a microscopic view depicts fine detail. Figure 1 illustrates a simple example of this distinction.
- *Macro/micro composition and reading*—a display composition that allows perception of both local detail and global structure. In such a display, fine details are discernible, but the details accumulate into larger coherent features, as in Figure 3.
- *Adaptive graphical display*—the adjustment of a display's graphical characteristics in response to data set size. The goal is to reveal as much detail as possible without visual complexity interfering with the perception of that detail. In Figure 3, for instance, the double cueing (height and color) of the vertical axis might be modified as data set size grows.
- *Display manipulation*—the interactive modification of a display, through techniques such as scrolling or zooming, to handle a large amount of data of varying detail. In Figure 1 (bottom), for example, scrolling or zooming along the time axis could be used to convey fine detail for long runs that would otherwise compress the time axis and lose detail.
- *Composite view*—a synthesis of two or more views into a single view that is intended to enhance visual relationships among the views and present more global information.[7] Examples include combining lower dimensional displays into a single higher dimensional display or taking time along a third axis to construct a three-dimensional display from a two-dimensional animation. As a simple illustration, Figure 1 (bottom) is essentially a composite of successive processor-state displays.

User perception and interaction

Successful visual performance tuning depends on a synergistic feedback loop between the user and the visualization tool. The tool produces images that the user interprets, while the user selects views and options to guide the tool in detecting and isolating performance bottlenecks. Important concepts in this category include

- *Perception and cognition*—the sensory development of an impression, awareness, or understanding of a phenomenon. Human visual perception can grasp patterns, distinguish variations, classify objects, and so on, through size, shape, color, and motion. For example, the use of color in Figure 1 (top) gives an overall impression of processor utilization that is easily perceived. A familiar example where shape conveys information is the Kiviat diagram.

- *Observing patterns*—the observation of spatial, temporal, or logical patterns of behavior, which often indicate important interrelationships between models and data. For example, a repetitive pattern over time is often related to a program's iterative loops. As an illustration, the bottom of Figure 4 shows a repetitive pattern that is easily grasped by the eye.
- *User interaction*—selections that users make regarding alternate views, levels of detail, and display parameters. Such interaction lets users customize the visualization for a given situation to enhance understanding. For example, in a processor-oriented display, users can select a particular layout or ordering of processors to make patterns and relationships more evident, or they can study one specific processor in detail.

Comparison

Comparisons and cross-correlations between related views or representations can provide much insight into behavioral characteristics and their causes. Several graphical techniques can be used for visual comparison:

- *Multiple views*—the visual presentation of data using multiple displays from different perspectives. Any single visualization or perspective can usually display only a portion of the relevant behavior. Viewing the same underlying phenomenon from diverse perspectives gives a more well-rounded impression and is more likely to yield useful insights.[8] For example, the top and bottom parts of Figure 1 give related but complementary views: One better indicates load balance, while the other better indicates concurrency. Still other views, say of communication or data accesses, could convey additional perspectives on the same underlying behavior.
- *Small multiples*—a series of images showing the same combination of variables indexed by changes in another variable, much like successive frames of a movie. Information slices are positioned so that the viewer can make comparisons at a glance. Animation is one example of this technique (indexing over time), but indexing can also be done by processor number, code version, problem size, machine size, and so on. Figure 3 illustrates this powerful technique.
- *Cross-execution views*—the visual comparison of performance information from program executions that may differ in various ways, such as in problem size or machine size. For example, Figure 5 (produced by ParaGraph[1]) shows a sequence of executions for successive program modifications.

Extraction of information

Several techniques enable visual extraction of useful information from a morass of data:

- *Reduction and filtering*—representing raw data by statistical summaries, such as maxima and minima, means, standard deviations, frequency distributions, and so on. This notion extends to graphical reduction, where a visual display conveys general trends rather than detailed behavior. In Figure 1 (top), for

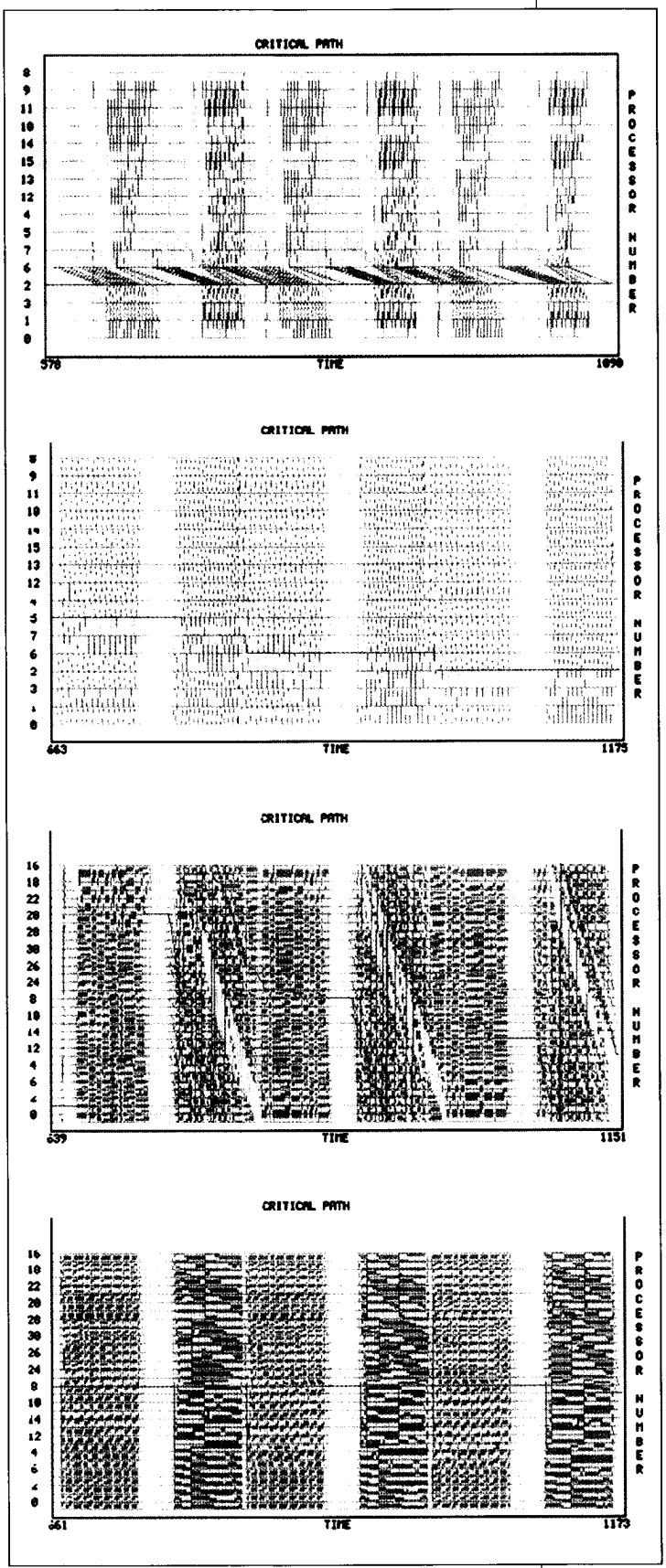


Figure 5. Viewing interprocessor communication in ParaGraph: (top to bottom) a series of critical-path displays shows successive improvements in parallel-program performance.

example, the data are reduced by the averaging of states over time.

- *Clustering*—multivariate statistical analysis and presentation techniques for grouping or categorizing related data points. The intent is to classify points or identify outliers in a multidimensional data space. Classical examples include scatterplots and frequency histograms.
- *Encoding and abstracting*—using graphical attributes such as color, shape, size, and spatial orientation or arrangement to convey information (for instance, additional dimensions). Such overloading can be easily abused, but when used appropriately, color coding and so forth can effectively increase a flat display's dimensionality. In Figure 3, for example, both color and spatial arrangement convey information.
- *Separating information*—visual differentiation among layers of information through color highlighting, foreground/background, and so forth. A good illustration of this technique is the highlighting of the critical path in Figure 5.

CASE STUDIES

We now illustrate, through a series of case studies, the practical application of the model we have proposed for performance visualization. (For further details and additional examples, see Heath, Malony, and Rover.[9])

Concurrency and communication in data-parallel computation

Our first example, discussed in detail by Rover and Wright,[10] represents the back substitution phase of a data-parallel program for solving a system of linear equations by Gaussian elimination. The system matrix is partitioned among the processors, and the owner-computes rule is used: Each program assignment statement is executed by the processor that owns the variable on the left-hand side, and any data used on the right-hand side that is owned by another processor must be sent to the left-hand-side owner before the statement is executed.

Our initial analysis abstraction is simply the processor states during program execution, while the visual abstraction is a time-series plot of the number of processors in each state. Scanning (scrolling, if needed) the display's time axis lets users perceive the state transitions and patterns, as shown in Figure 4 (top), which reveals that processor utilization is poor. Suspecting that this inefficiency may stem from communication of nonlocal data, we select a view that depicts the message-passing among the processors—specifically, the space-time diagram shown in Figure 4 (middle). A clear pattern emerges, with repetitive communication rounds among processor groups.

To investigate how these groups relate to their locations, we select a processor-oriented view whose spatial arrangement of the processors reflects their logical configuration in a two-dimensional grid for the purpose of partitioning the matrix. Tying the message-passing and processor-oriented views together, Figure 4 (bottom) represents a display showing processor states and communication animated over time. From the combination of shape and color coding, we see that only one processor row (corresponding to the processor groups we saw in the space-time diagram) is busy at a time, while all other processors await remote data. As the animation shows, data are sent from a processor row to the row above it, and the busy processor row cycles upward. Thus, concurrency is limited to one processor row at a time, which explains the poor utilization that we noticed initially.

The critical path's stability across multiple executions of the program is especially significant.

Access patterns for data distributions

Our second example involves data-access patterns in parallel-programming languages, such as High-Performance Fortran (HPF) or parallel C++, that incorporate data-distribution semantics. Interprocessor communication in such languages is implicitly determined by the data distribution; the distribution selection constitutes the programmer's main control over parallel efficiency. An appropriate analysis abstraction, therefore, is the proportion of local versus remote data accesses required to support a particular data distribution.

For an application involving Gaussian elimination on a matrix, the relevant data structure is a two-dimensional array supporting an effective performance view that relates easily to the application program. The visual abstraction represents the data structure as a surface whose height and color are determined by the proportion of local data accesses for the corresponding position in the two-dimensional array (Figure 3). Color enhances viewer perception of the differences in data accesses within a single image and helps unify the series of images. The surface changes could be animated over time, but perhaps even more effective is a series of snapshots (small multiples) that convey the changing data-access patterns as the algorithm progresses.

Critical paths in parallel computation

The critical path is the longest serial thread, or chain of dependencies, running through a parallel-program execution. It is an important performance analysis abstraction, because we cannot reduce the program's execution time without shortening the critical path; hence, it is a potential place for bottlenecks. The critical path's stability across multiple executions of the program is especially significant, since it may reveal the presence or absence of a systematic bias in the execution. By itself, however, critical-path stability does not tell us whether such a bias is good or bad, so it should be augmented by other views.

For parallel programs based on message passing, an appropriate visual abstraction for depicting the critical path is a minor modification of a space-time diagram, since data dependencies are satisfied by interprocessor communications. Figure 5 shows the critical paths (highlighted in the figure) for a sequence of successive improvements in a parallel program for solving shallow-water equations on a sphere using a spectral transform method.[11] Time has been rescaled in each instance, so the time scale shown

does not reflect the actual performance improvement, which is more than a factor of three. Actually, the programmer made these improvements by analyzing program behavior using utilization and task displays, but it is nevertheless instructive to observe the critical paths' resulting behavior.

In the initial implementation (Figure 5, top), the performance is poor because of a substantial load imbalance, and the critical path, not surprisingly, stays with a single processor (the most heavily loaded one). When the load balance is improved (Figure 5, middle top), the critical path begins to cover more processors, as no single processor is now the bottleneck. Further improvements spread the critical path even further (Figure 5, middle bottom). This appears to support the intuitive notion that a well-balanced algorithm should produce a somewhat random critical path because of slight timing vagaries in the nearly equal tasks. After the final improvement, however, the critical path once again settles mainly on one processor (Figure 5, bottom). The explanation is that this implementation uses a carefully pipelined, ring-oriented algorithm. All previous load balance and communication anomalies have been removed, so that the behavior is now quite regular, with the trailing processor in the pipeline consistently staying on the critical path.

Scientific data can also reflect discrete events having intricate temporal precedence relationships, such as elementary particle tracks from an accelerator.

SCIENTIFIC VISUALIZATION AND PERFORMANCE VISUALIZATION

Although there are differences, scientific visualization and performance visualization have similar aims: to gain insight into underlying phenomena by graphically depicting data.

Nevertheless, some argue that scientific visualization has the advantage of representing real phenomena, while performance visualization handles abstractions and artificial objects. However, this is only partially true and is based on a rather narrow view of scientific visualization. Obviously, many visualizations do involve the depiction of some continuous physical quantities, such as temperature or pressure, as a function of some continuous variables, such as space or time coordinates. And the graphical image presented, typically represented by lines and surfaces in Euclidean space, does correspond directly to some physical system. But many other types of scientific data that involve abstract entities or discrete objects bear little relation to any intuitive image of underlying reality.

For example, a dynamic system's behavior is often best understood in terms of phase space rather than the ordinary space-time in which the model is formulated. Phase space is an abstraction that scientists understand. When presented with a graphical image of it, they know what patterns (orbits and so forth) to look for. These patterns would likely be less obvious in an ordinary graphical presentation of the same data in space-time. The use of such abstractions to interpret and represent data is ubiquitous throughout science. Even time representation by one spatial axis is an abstraction in this sense. Thus, it is not the direct representation of reality that gives scientific visualization its power, but rather the proper matching of a graphical image to the scientist's abstraction, regardless of how far that picture may be from ordinary reality.

Consider some typical uses of graphics, say in the statistical analysis of data. A simple scatterplot of the residuals in fitting a model to data can reveal outliers or systematic bias that might go undetected by sophisticated analytical or computational methods. Such a plot has no direct correlation with "reality." It does not depend on the nature of the quantities represented by the data or the model. But it is nevertheless a powerful graphical technique that takes full advantage of the human eye's ability to spot patterns. Another example of this type is the use of purely abstract graphical plots to detect nonrandom patterns in the output of a (supposedly) random number generator.

A second argument is that performance data are fundamentally different from typical scientific data because of the discrete nature of the performance events and their complex, logical interrelationships. Again, this would be true if we took a narrow view of scientific data as basically the solution of a field equation. However, in a broader sense, scientific data involve discrete entities—genome sequences, demographic surveys, and so on. Scientific data can also reflect discrete events having intricate temporal precedence relationships, such as elementary particle tracks from an accelerator. A common feature in many of these cases is that they tend to be pattern recognition problems where qualitative results are more important than quantitative results. This makes them even more ripe for graphical presentation than more conventional scientific modeling of physical phenomena, since such patterns are often best grasped visually. Since performance visualization has much the same flavor, both in its goals and in the data with which it typically works, perhaps we should be looking here for useful analogies and graphical ideas.

This is not to say that performance data and visualization have nothing in common with conventional scientific data and visualization, where contours, surfaces, and so on depict field quantities. Performance data are almost invariably time dependent and often involve space-like quantities (memory addresses, loop and array indices, process and processor identifiers, and so on). This lets many standard tools of conventional scientific visualization be effectively applied in visualizing performance data.

Hence, experience gained and lessons learned in scientific visualization have a direct benefit in developing and applying performance visualization technology. Specifically,

- abstraction is a powerful and ubiquitous tool throughout science, and one key to success in visualization is the appropriate matching of graphical images to these abstractions;
- scientific data come in many varieties and often present pattern recognition problems that may have much in common with performance analysis and visualization;
- simple graphical techniques can be powerful in yield-

ing insights, but the entities represented may bear little or no direct relation to any physical entities; and
- we should use the full range of tools and techniques for scientific visualization in seeking new ideas and paradigms for performance visualization.

DESPITE TECHNOLOGICAL ADVANCES in performance visualization design, developing useful parallel performance visualizations is still challenging. Miller[12] addresses this challenge, listing criteria for good visualization. These criteria, like the concepts we have discussed, help define general requirements to guide the visualization designer in creating useful visual displays. Of course, in enhancing the interpretation of a visualization, we must also consider the semantic context—particularly with respect to the user's conceptual model. A visualization that is meaningful to one person may not be meaningful to another.

The performance visualization model we have presented emphasizes the need for integrating performance evaluation models with performance displays to provide a foundation for developing useful visual abstractions. But the model also acknowledges the need for user involvement—in design, interaction, and evaluation—from which true insight is ultimately achieved. ▮

References

1. M. Heath and J. Etheridge, "Visualizing the Performance of Parallel Programs," *IEEE Software*, Vol. 8, No. 5, Sept. 1991, pp. 29-39.
2. P. Keller and M. Keller, *Visual Cues: Practical Data Visualization*, IEEE Press, Piscataway, N.J., 1993.
3. E. Tufte, *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, Conn., 1983.
4. E. Tufte, *Envisioning Information*, Graphics Press, Cheshire, Conn., 1990.
5. S. Hackstadt and A. Malony, "Next-Generation Parallel Performance Visualization: A Prototyping Environment for Visualization Development," Tech. Report CIS-TR-93-23, Dept. of Computer and Information Science, Univ. of Oregon, Eugene, Ore., 1993.
6. J. Yan et al., *The Automated Instrumentation and Monitoring System (AIMS) Reference Manual*, Report 108795, NASA Ames Research Center, Moffett Field, Calif., 1993.
7. A. Couch, "Categories and Context in Scalable Execution Visualization," *J. Parallel and Distributed Computing*, Vol. 18, No. 2, June 1993, pp. 195-204.
8. T. LeBlanc, J. Mellor-Crummey, and R. Fowler, "Analyzing Parallel-Program Executions Using Multiple Views," *J. Parallel and Distributed Computing*, Vol. 9, No. 2, June 1990, pp. 203-217.
9. M. Heath, A. Malony, and D. Rover, "Parallel Performance Visualization: From Practice to Theory," *IEEE Parallel and Distributed Technology*, Vol. 3, No. 4, Winter 1995.
10. D. Rover and C. Wright, "Visualizing the Performance of SPMD and Data-Parallel Programs," *J. Parallel and Distributed Computing*, Vol. 18, No. 2, June 1993, pp. 129-146.
11. P.H. Worley and J.B. Drake, "Parallelizing the Spectral Transform Method," *Concurrency: Practice and Experience*, Vol. 4, No. 4, June 1992, pp. 269-291.
12. B. Miller, "What to Draw? When to Draw? An Essay on Parallel-Program Visualization," *J. Parallel and Distributed Computing*, Vol. 18, No. 2, June 1993, pp. 265-269.

Michael T. Heath *is a professor in the Department of Computer Science and a senior research scientist at the National Center for Supercomputing Applications at the University of Illinois, Urbana-Champaign. His research interests include large-scale scientific computing on parallel computers, with specific interests in sparse matrix computations and performance visualization.*

Heath received a BS degree in mathematics from the University of Kentucky, an MS degree in mathematics from the University of Tennessee, and a PhD degree in computer science from Stanford University. Heath is a member of the scientific advisory and review panels for high-performance computing at Argonne, Los Alamos, and Oak Ridge National Laboratories and is on the panel of judges for the Gordon Bell Prize for parallel performance. He is an editor of the SIAM Review, *the* SIAM Journal on Scientific Computing, *and the* International Journal of Supercomputer Applications.

Allen D. Malony *is an assistant professor in the Department of Computer and Information Science at the University of Oregon. His research interests focus on performance evaluation for parallel and high-speed computer systems, including performance measurement, analysis, visualization, and modeling, as well as parallel-programming environments.*

Malony received BS and MS degrees in computer science from the University of California, Los Angeles, in 1980 and 1982, respectively, and a PhD degree from the University of Illinois, Urbana-Champaign, in 1990. He received a National Science Foundation (NSF) National Young Investigator Award in 1994.

Diane T. Rover *is an assistant professor in the Department of Electrical Engineering at Michigan State University. Her research interests include integrated program development and performance environments for parallel and distributed systems, instrumentation systems, performance visualization, embedded real-time system analysis, and reconfigurable hardware.*

Rover received a BS degree in computer science in 1984, and MS and PhD degrees in computer engineering in 1986 and 1989, respectively, all from Iowa State University. She received an R&D 100 Award in 1991 for the development of the Slalom benchmark, a MasPar Challenge Award in 1994, and an MSU College of Engineering Withrow Teaching Excellence Award in 1994. Rover is a member of the IEEE Computer Society, the ACM, and the American Society for Engineering Education (ASEE).

Readers can contact Heath at the Department of Computer Science, 2304 Digital Computer Laboratory, University of Illinois, 1304 West Springfield Ave., Urbana, IL 61801-2987; e-mail heath@cs.uiuc.edu.