PerfVisS: A Performance Visualizer for High Performance Fortran Programs on Workstation Farms

Gudrun J. Klinker
Cambridge Research Lab, Digital Equipment Corporation
I-Yu Chen
Mercury Computer

Digital Equipment Corporation
Cambridge Research Lab

CRL 94/4 June 30, 1994

Abstract

Writing efficient code for parallel processors is still a rather complex and little understood task. Tools to gather and analyze performance information during program execution are essential for understanding the reasons for inefficient executions. Current performance visualization systems provide only a limited set of capabilities, hardwired into huge, monolithic programs. Programmers need a very flexible environment in which they can mix and match different performance visualization tools.

Research in scientific visualization has developed several environments to visualize, explore and analyze large quantities of data. Reusing these capabilities to visualize MIMD performance data significantly helps in the development of a performance profiler. However, scientific visualization methods have to be altered appropriately to address the non-geometric nature of performance data.

This paper presents a profiler, PerfVisS, which builds upon existing Telecollaborative Data Exploration (TDE) technology. PerfVisS provides several linked views of the performance data, both for gaining a general overview of the program performance and for detailed data and code inspection. It also allows users to adapt the system to their own viewing preferences, using the AVS visual programming interface. Furthermore, it provides tools for telecollaborative performance tuning, allowing programmers to share their views with colleagues at remote sites. PerfVisS runs on Alpha AXP™ workstation farms but could be used as well on other similar MIMD architectures such as symmetric multiprocessors (SMPs).

1 Introduction

Writing efficient code for parallel processors is still a rather complex and little understood task. On MIMD architectures, programmers have to decide how to design a collection of computationally balanced programs that optimize the use of computing resources across all processors. Tools to gather and analyze performance information during or after program execution are essential for understanding the reasons for inefficient executions. But, performance gathering tools generate large amounts of data, requiring sophisticated data visualization tools.

1.1 Existing performance visualization systems

Several performance visualization systems, such as MulTVision [2], ParaGraph [3, 4], Upshot [5], and DIVIDE [14] have recently emerged. They use event-based trace data, helping programmers analyze specific aspects of their programs, such as the average and variance of the communication buffer length [14], process state changes over time [2, 3, 4], and user-selectable events [5]. Many different data display metaphors have been explored; some provide static views of the data [2, 5] while others present dynamically changing views, indicating the event-based information changing over time [3, 4, 14].

While these systems are quite sophisticated, they provide only limited capabilities, hardwired into huge, monolithic visualization programs. Since it is still unclear what constitutes an optimal set of performance visualization tools, programmers need a very flexible environment in which they can mix and match different performance visualization tools. Furthermore, performance visualization also needs to help programmers analyze the data, using appropriate data filtering tools, as suggested by Waheed and Rover [16].

1.2 Use of scientific visualization packages for performance visualization

Several scientific visualization environments offer a large degree of flexibility, as well as a growing collection of data presentation and data analysis tools for large quantities of – typically geometric – data [7, 12, 13, 15]. Reusing these capabilities to visualize MIMD performance data significantly helps in the development of a performance profiler.

Yet, several problems have to be addressed. How can performance profiles be represented in data formats supported by scientific visualization tools? How can visualization tools be extended to appropriately manipulate non-geometric information for which the horizontal and vertical dimensions no longer adhere to the conventional Eucledian interpretation but rather represent dimensions like time, permutable lists of processors, or lines of source code. Another question is adequate data reduction for generation of suitable overviews of the data.

Typical geometric schemes, such as subsampling or linear interpolation, are no longer applicable because the semantic relationships between neighboring measurements have to be taken into account.

1.3 PerfVisS

We have developed a performance profiling system, PerfVisS, for visualizing and exploring parallel executions of programs on workstation farms. PerfVisS provides several views of the data at different levels of granularity. It shows overviews of the data which allow programmers to navigate through the vast amount of information from a higher-level perspective, while providing some guidance for finding critical code segments. PerfVisS also provides several detailed views of selected segments of the data: programmers can see image-based and geometric data renditions, as well as text-based presentations, including print outs of individual performance data values, lists of source code and assembler code, and ranked performance information. All views are aligned and linked such that user interaction with one window automatically updates the views in all other windows.

PerfVisS uses existing scientific visualization and data exploration technology, namely AVS [15] and TDE [7]. Due to the visual programming interface of AVS, new views can be integrated easily, making this an ideal prototyping and research environment for determining suitable visualization metaphors for performance data.

PerfVisS especially benefits from the telecollaborative data exploration capabilities of TDE, allowing programmers to discuss their performance data with remote colleagues or application centers of computer vendors, each being able to interact with the data. For today's international corporations, we consider telecollaboration to become an essential component of program development efforts, blurring the distinctions between collaborating programmers working in separate offices, sites, states, or countries.

1.4 Application domain: pc-sampled High Performance Fortran (HPF) programs on workstation farms

PerfVisS is designed to help programmers understand executions of High Performance Fortran programs running on Alpha AXP™ workstation farms. It could also be used for other similar MIMD architectures, such as symmetric multiprocessors (SMPs).

Most of today's workstation farms or clusters run single-threaded batch jobs. A greater pay off can be expected by farms used as "degenerate MPPs," with parallel processing being the technical underpinning. Because farms reach a wider audience than MPP systems, the need for a portable, high level, and stable programming paradigm is crucial. High Performance Fortran (HPF), coupled with compilers, debuggers, performance profilers and visualizers, has the potential to offer such a programming paradigm [1].

HPF [6, 10, 11] provides language features allowing application programmers flexibility in developing or migrating computationally intensive applications to high performance computing environments while at the same time allowing efficient implementation on a wide variety of platforms. HPF is Fortran 90, augmented by a small number of de facto industry standard extensions in four areas.

- Data layout and placement directives, such as the DISTRIBUTE directive applied to arrays.
- Parallel statements and directives, such as the FORALL construct, which is an element-at-a-time generalization of data parallel array operations.
- Intrinsic and library procedures.
- The EXTRINSIC capability. EXTRINSIC procedures define an explicit interface to procedures written in other paradigms, such as SPMD with explicit message-passing.

In our applications, the same HPF-code executes on all processors, operating on different parts of large arrays. Performance data is collected by sampling the program counters of all processors at regular intervals. After program execution, the collected data is projected into histograms, indicating how much time each processor spent executing each instruction. PerfVisS allows programmers to explore these histograms.

2 Presentation of performance profiles

Figures 1 through 5 show the performance profiles generated by PerfVisS. In a profiling session, the windows are displayed on a workstation screen in a predefined layout. Users can change the layout by resizing and repositioning windows. They can also duplicate and redisplay windows on other screens to discuss the data with colleagues. This section describes the data presentations. The next section describes how PerfVisS is implemented in TDE, our AVS-based Telecollaborative Data Exploration environment.

2.1 Data mapping

When users develop and run programs, a lot of general program information, as well as execution-specific performance information, is generated in many different files. In particular, the HPF performance profiler for Alpha AXP™ workstation farms generates a profile, describing the execution frequency ("tick count") of every line, l, of source code on every processor, c.

2.1.1 A detailed view of the performance data

PerfVisS begins by loading the performance profile into a two-dimensional array. Along the horizontal axis, it enumerates all processors, c. The vertical axis represents all lines, l, of source code. The result is a long, slim array, representing the performance data per line and per processor. Figure 1a shows a short segment of the array for the execution of a program running on 4 processors.

In order to help programmers quickly determine the busiest "hot spots" in a program segment, we have color-encoded the performance data using a heat map. Inactive (cold) code segments are shown in blue and green,[1] busy (hot) code segments in red. The colors help programmers find the hot spots quickly, as well as visualize the performance difference between the processors. Overlaid on the color picture are the actual tick counts. These printed numbers allow programmers to inspect the performance data in detail when necessary. Programmers can display the information normalized (i.e., only the hottest spot is shown in bright red) or unnormalized.

2.1.2 Viewing the source code and the assembler code

Aligned with the detailed view of the performance data, PerfVisS shows a window with the source code (Figure 1b). Optionally, programmers can also inspect the assembler code by selecting a source line in the detailed view (Figure 1a) with the mouse. The assembler code from a user-specified number of lines is then shown in a separate window (Figure 2).

2.1.3 Slowest and fastest processor performance

When many parallel processors are used for the computation, programmers need to visualize the performance variation across all processors. PerfVisS sorts processor performance per line of code and computes the average and variance values. Figure 1c shows the minimal and maximal tick counts. The dark horizontal bars indicate minimal tick counts per source line; the light bars indicate the difference between minimal and maximal tick counts. The counts and the associated processor numbers are also printed next to the bars.

2.1.4 An alternate view: performance data as a three-dimensional profile

In addition to rendering the performance data as a heat map, we display it as a three-dimensional surface, as shown in Figure 3. In this presentation style, the busiest source lines stand out as high peaks in a mountainous terrain. When the surface is rotated upwards and sidewards, it resembles sets of two-dimensional

[1]As a special case, we display comments and other non-operative lines with zero tick counts in black.

performance profile plots programmers may already be familiar with. Furthermore, it may be the only understandable presentation for color blind people. On the other hand, it cannot be aligned as succinctly with printed tick counts and listings of source code: a vertically aligned projection of the profile (similar to the sorted performance data of Figure 1c) needs a lot of space to adequately lay out all printed tick counts.

In conclusion, the color-encoded heat map and the three-dimensional profile complement each other, aiding programmers through different viewing metaphors. Our telecollaborative data exploration environment, TDE, allows us to easily generate such alternate views of information. We expect this to be an important component in study of the suitability of different tools for performance visualization.

2.2 Data reduction for non-scientific data

The displays shown so far allow programmers to study performance data in great detail. Yet, they do not help gain a general overview of the collected information. Programmers need additional visual aids to determine quickly which program areas are most critical so that they can dive into a detailed investigation of such trouble spots. Systems like DIVIDE [14] provide sophisticated slider widgets[2] mimicking the jog-shuttle metaphor of VCRs to help users find the view they need. These widgets do not help get an overview of the data content; they merely provide movement control, asking users to "blindly" find the most relevant data sections.

Scientific visualization applications, on the other hand, do provide overviews. The overviews are typically generated by subsampling or interpolating between several elements of large arrays, generating data pyramids with varying resolution. These techniques cannot be applied directly in the performance visualization context because the performance information is not geometrically coherent: we might easily miss important hot spots. Furthermore, the performance array is disproportionately long, requiring that data compression along the horizontal and vertical dimension be treated separately.

We provide programmers with two tools to gain an overall impression of their performance data.

2.2.1 Generating an overview

We exploit several properties of performance data when we reduce its size to generate an overview.

First, blank lines can be removed without distorting the overall information content. Our system thus skips code segments with more than a user-specified number of consecutive empty lines (i.e.: tick counts = 0).

[2]along the time axis rather than source line numbers

Second, profile peaks are much more important than profile minima. We can adequately represent bins of consecutive source lines by a single one, if we maintain the maximal tick counts. In the compressed representation, programmers are still alerted to the hot spots, and they can determine the exact structure by generating a detailed view of the area around the hot spot. However, this compressed view does not maintain an adequate presentation of the extent of a hot spot over several consecutive lines, such as the body of a loop. Another compression technique would be to accumulate the tick counts across several source lines. This rendering, however, reduces the visibility of small, high peaks. Our system allows programmers to switch between these data compression schemes.

Third, we exploit the slim nature of the performance array, cutting it into several shorter strips which we then display side-by-side – in analogy to page breaks in printed source code. This heuristic works very well for performance data from a small number of parallel processors.

We have not yet provided means to also compress the horizontal dimension, should the number of processors become large. Tools could use the ranked performance data per source line, as described in section 2.1.3 (Figure 1c), to show only a selected number of ranks.

Figure 4 shows an overview of the performance data gathered from a "spike" program designed to test the functionality of an HPF compiler under development. The code initially consists of 94824 source lines (including linked-in libraries). To generate the overview picture, we have skipped code segments with more than 50 consecutive empty lines[3]. We have compressed bins of 4 source lines into one line, cut the array into 14 strips of 80 lines each, and used the heat map encoding scheme. Overlaid on the data are a yellow rectangle (eighth strip) surrounding the code segment of the busiest procedure (see below), and a golden rectangle (leftmost strip) indicating the selected data for the detailed view. Programmers select a detailed view in the overview window with the mouse to dive into a detailed analysis of a hot spot. Our system then automatically focuses all detailed data displays on the selected code segment (Figures 1 and 3).

2.2.2 Ranked procedure performance

From traditional, text-based profiling tools, programmers are used to work with lists of ranked procedure performance. To ease programmers into understanding the color-encoded views, and to impose more structure on such views, we also provide a ranked list of the top n procedures (Figure 5).

When one of the routines is selected with the mouse, the corresponding code segment in the overview window is automatically surrounded with a yellow rectangle. Programmers can also request to see the outlines of all listed procedures by clicking on the title row.

[3]Many linked-in library routines are not executed at all and can thus be skipped

Conversely, programmers can determine for any code segment in the overview window what the associated procedure name is. When a source line is selected in the overview window, the corresponding procedure name is highlighted in the ranked procedure listing. If the procedure is not among the top n procedures, it is listed at the bottom of the window.

3 Prototype Implementation

We have implemented our performance profiling system using TDE, a Telecollaborative Data Exploration environment [7].

3.1 TDE

TDE uses the visual programming interface and the runtime data flow executive of AVS [15]. TDE extends AVS with very sophisticated data exploration and telecollaboration capabilities. It also provides the tools to register geometric data with array-based information, overlaying text or drawings on images.

We use the following capabilities of TDE for PerfVisS:

- Flexible data presentations
  Figures 1 through 5 have shown several different presentations of the performance data, including color-encoded heat maps, three-dimensional profiles, and printed data values. Ranked processor performance per source line was shown as bar graphs (Figure 1c). Geometric drawings, such as rectangles and text, are overlaid on the color-encoded performance data to amplify the visual expressiveness of each display.

- Data probing
  TDE provides data probing capabilities, allowing users to determine and select regions of interest in the overview window (Figure 4) and the ranked procedure window (Figure 5). The three-dimensional profile (Figure 3) can be rotated interactively. Assembler instruction code can be obtained by probing a line of the source code in the detailed window (Figure 1a).

- Cursor linking
  When users interact with one of the windows, all other windows are automatically updated accordingly. To maintain linked cursors between several, semantically related windows, TDE provides a *log record* [7, 8]. The log record establishes geometric consistency between several views even when they have been geometrically transformed. Using the log record, performance visualization modules provide the semantic translation between different views.

- Geographically distributed windows
  Exploiting the client-server capabilities of X, TDE provides a *window mi-*

gration capability to send windows to any display [9]. Accordingly, users are not limited to a single screen but can use all screen space at their disposition.

Users can also send windows to colleagues or to application centers of computer vendors to discuss alternate program parallelization approaches. During such telecollaboration, users can duplicate windows and link cursors, using the visual programming interface of AVS. The duplicates can then be forwarded to consultants at remote sites, due to TDE's window migration tools. The result is a fully duplicated performance visualization environment, providing full explorative capabilities to each viewer, while ensuring consistent views between all sites. The remote display sites do not require any special purpose hardware or software installations, such as TDE or AVS – just an X-server.

- Customization
  Since PerfVisS uses the visual programming interface of AVS, we are able to quickly prototype various potential performance visualization interfaces, studying their suitability for optimizing parallel programs. The visual programming interface enforces a high degree of modularity. We are able to reuse many modules originally developed for scientific visualization applications. Most of the modules that were developed specifically for the performance visualization application read in files in specific data formats or reduce the performance data to generate overviews.

  The embedded data flow scheme allows us – as well as users – to spontaneously include new visualizations and data analysis modules [16] into PerfVisS, merely by defining data flow connections to new modules. We can establish links to such new, semantically related, presentations by providing the appropriate semantic cursor translation algorithms.

3.2 Use of TDE for performance visualization

Figure 6 shows the AVS data flow network for PerfVisS, constructed out of TDE modules and a few PSE modules developed specifically for the HPF Parallel Software Environment to read and process the performance data. The data flow network consists of seven major blocks of modules, as shown in the schematic overview of Figure 7. Each block, except for A, corresponds to a displayed window of the profiler.

When the performance data is read into the network (block A), it flows to block C. Several non-geometric transformation are applied to the data: blank lines are skipped, the data is reduced by maximizing tick counts across several lines of code per bin, and the data is then cut into parallel vertical strips. The data then is normalized and color-encoded, before it is finally displayed.

Geometric descriptions of rectangles coming from the ranked procedure window (block B) and the detailed view (block D) are overlaid on the color-encoded overview image to outline selected code segments. Blocks *B* and *D* provide the rectangles in the original coordinate frame. Module *PSE Wrap Rectangles* of block *C* applies the same non-geometric transformations to the rectangles that are applied to the performance data, compressing the rectangles and cutting them into several pieces, if the outlined code segments spread across several strips. Module *PSE Unwrap Reduced Line Pos* performs the inverse transformation on a cursor position from the overview window, mapping it back to the original coordinate frame.

The selected and transformed cursor position flows from block *C* to *B* where it is used to highlight the corresponding procedure name in yellow. The correct procedure is determined by reading through a file of procedure descriptions, determining which procedure includes the selected source line. A rectangle outlining the associated code segment is generated and sent back to block *C*.

Block *D* uses the selected and transformed cursor position from the overview window to crop a code segment from the performance data, generating a new, much shorter array. Users can specify interactively how many source lines to crop. The code segment is displayed as a heat map with overlaid printed numbers, generated by the *TDE draw numbers* module and the *TDE change color* module. Should we decide to present more information about the code segment or to render it differently we merely have to include the appropriate new modules at this place into the data flow network.

The cropped code segment flows to several other blocks, *E*, *F*, and *G*. These blocks display the source code, the three-dimensional profile, and the sorted processor view.

Figure 8 shows the data flow network of Figure 6, adjusted for telecollaborative use between two sites. All *TDE display data* modules have been duplicated, with identical input flowing into each pair. The output from each pair is multiplexed so that user input from either window is forwarded to subsequent modules for further interpretation. The result is a fully duplicated performance visualization environment, providing full explorative capabilities to two viewers, while ensuring consistent views at both sites. The remote display site does not require any special purpose hardware or software – just an X-server.

4 Summary

In this paper, we have presented PerfVisS, a performance visualization system for pc-sampled HPF programs running on Alpha AXP™ workstation farms or similar MIMD architectures such as SMPs.

The system builds upon the power of existing scientific visualization software, AVS and TDE. As a consequence, we were able to rapidly develop a prototype. Due to the data exploration capabilities of TDE, we are able to provide many different views of the data, including image-based heat maps, geometric three-dimensional profiles, and text-based presentations. Information is shown at several levels of granularity, allowing users to gain both a general overview and a detailed look. Since TDE provides a framework to establish linked cursors between several, semantically related windows, all views in PerfVisS are linked so that user interaction with one window automatically updates all other windows. Due to TDE's telecollaborative capabilities, PerfVisS users can include colleagues at remote sites into their performance tuning tasks.

Applying scientific visualization software to non-geometric data was a valuable learning experience. Visualization concepts had to be carefully evaluated and extended to warrant their proper use on data where array elements can be permuted along some feature axes, and where the relationships between neighboring array elements are non-geometric.

PerfVisS is a stepping stone towards very general performance visualization environments. Future plans cover a variety of extensions. We intend to integrate the system seamlessly with the HPF Parallel Software Environment by connecting profiling, debugging, and editing tools. We also plan to provide visualizations of event-based performance data. Finally, we are developing a cascade of views at many different levels of granularity so that very large performance data sets can be presented with varying amounts of detail.

Due to the rapid prototyping facilities and the large collection of data exploration tools in TDE and AVS, we have been able to develop the PerfVisS prototype easily and quickly. In its current form, PerfVisS has already proven to be an invaluable tool in testing a variety of methaphors for performance visualization. We thus expect it to lead us towards sophisticated, tele-collaborative performance exploration environments.

Acknowledgments

Many people have helped develop PerfVisS. During many discussions and live demonstrations of PerfVisS, Chuck Wan, John Clark, David LaFrance Linden, Tom Stones, and Brendan Smith of Digital's High Performance Computing Group (HPCG) have provided valuable insights regarding suitable data presentation formats for HPF-programms. Bert Halstead from the Cambridge Research Lab (CRL) and Marco Annaratone from HPCG were instrumental in establishing the link between research in scientific visualization and performance visualization. We are also grateful to Mark Brown, the director of CRL, and Ingrid Carlbom, the director of visualization research at CRL, for their support.

References

[1] M. Annaratone, D. Loveman, and C. Offner. High performance fortran on workstation farms. In *Proc. IPPS*, Cancun, Mexico, April 1994. IEEE.

[2] R. Halstead, D. Kranz, and P. Sobalvarro. MulTVision: A tool for visualizing parallel program executions. In R. Halstead and T. Ito, editors, *Parallel Symbolic Computing: Languages, Systems, and Applications*, pages 183–204. Springer-Verlag Lecture Notes in Computer Science 748, 1993.

[3] M. Heath and J. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, 8(5):29–39, September 1991.

[4] M. Heath and J. Etheridge. Visualizing the performance of parallel programs. Technical Report ORNL/TM-11813, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1991.

[5] V. Herrarte and E. Lusk. Studying parallel program behavior with Upshot. Technical Report ANL-91/15, Argonne National Laboratory, Argonne, Illinois, 1991.

[6] High Performance Fortran Forum. High Performance Fortran language specification, version 1.0,. Technical Report CRPC-TR92225, Center for Research on Parallel Computation, Rice University, Houston, TX, May (revised) 1993. Also appeared in a special issue of Scientific Programming, vol 2 (1 and 2), John Wiley & Sons, Spring and Summer 1993.

[7] G.J. Klinker. An environment for telecollaborative data exploration. In *Proc. Visualization '93*, pages 110–117, San Jose, CA, Oct 1993. IEEE Computer Society Press.

[8] G.J. Klinker. Interactive data exploration and telecollaboration in biomedicine using AVS. In *Proc. of the 2nd Int. AVS User Group Conference*, Walt Disney World Dolphin, FL, May 24-26 1993.

[9] G.J. Klinker, I. Carlbom, W. Hsu, and D. Terzopoulos. Scientific data exploration meets telecollaboration. In *Submitted to the 2. ACM International Conference on Multimedia*, San Francisco, CA, Oct. 15-20 1994. ACM Press.

[10] C.H. Koelbel, D.B. Loveman, R.S. Schreiber, G.L Steele, and M.E. Zosel. *The High Performance Fortran Handbook*. MIT Press, 1993.

[11] D.B. Loveman. High performance fortran. *IEEE Parallel & Distributed Technology*, 1(1), February 1993.

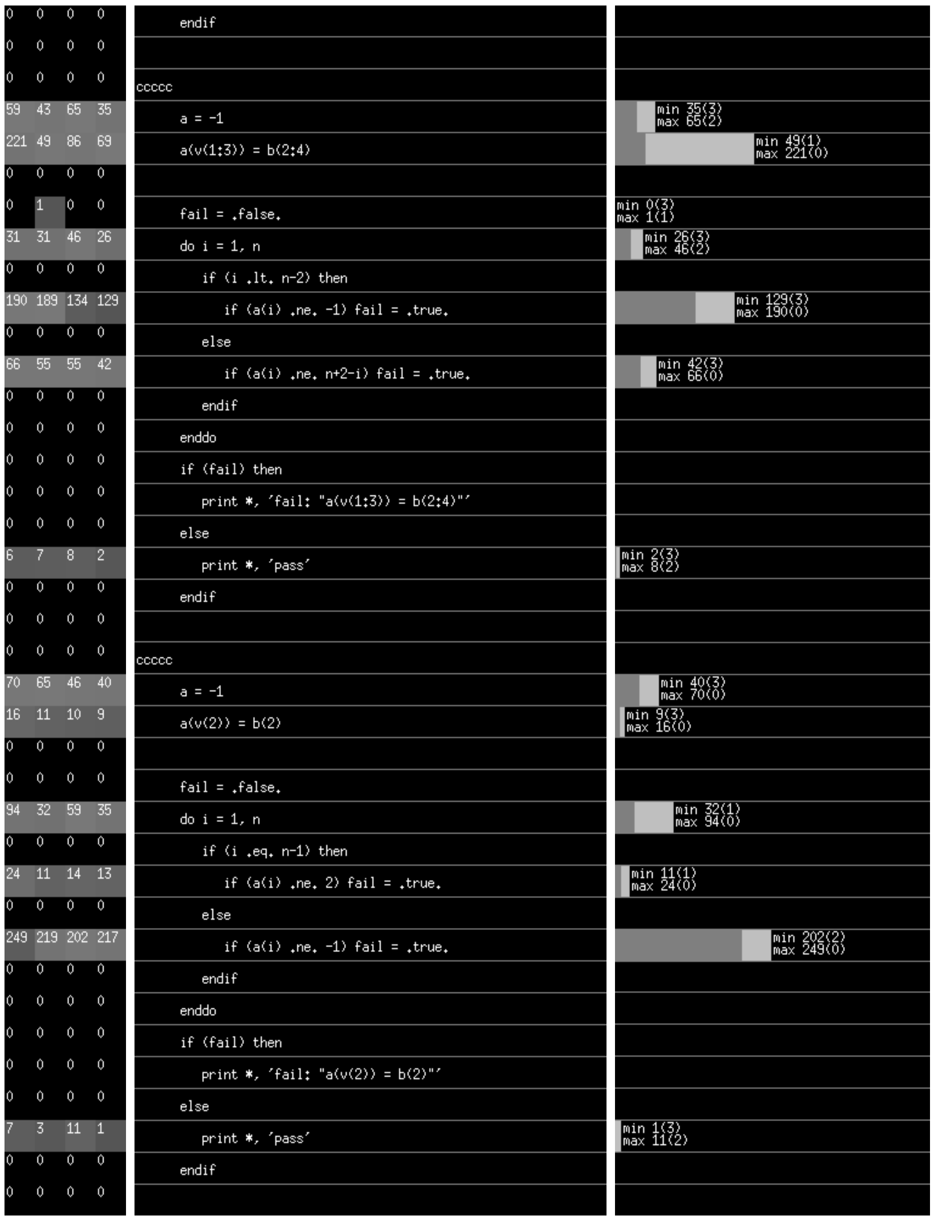
[12] B. Lucas, G.D. Abram, D.A. Epstein, D.L Gresh, and K.P. McAuliffe. An architecture for a scientific visualization system. In *Proc. of Visualization '92*, pages 107–114, Boston, MA, October 1992. IEEE Computer Society Press.

[13] P.J. Mercurio. Khoros. *PIXEL*, 3(1):28–33, 1992.

[14] T.M. Morrow and S. Ghosh. DIVIDE: Distributed visual display of the execution of asynchronous, distributed algorithms on loosely-coupled parallel processors. In *Proc. Visualization '93*, pages 166–173, San Jose, CA, Oct 1993. IEEE Computer Society Press.

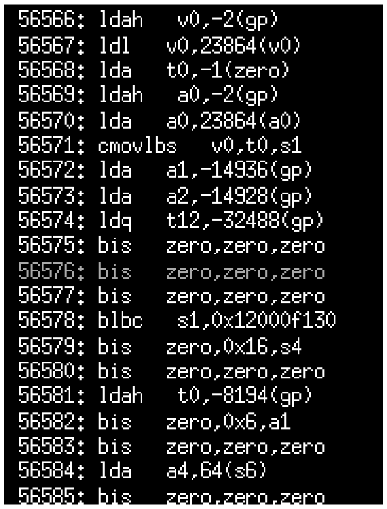
[15] C. Upson, T. Faulhaber Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. The Application Visualization System: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, July 1989.

[16] A. Waheed and D.T. Rover. Performance visualization of parallel programs. In *Proc. Visualization '93*, pages 174–181, San Jose, CA, Oct 1993. IEEE Computer Society Press.



```
0   0   0   0       endif
0   0   0   0
0   0   0   0    ccccc
59  43  65  35      a = -1                                  min 35(3)  max 65(2)
221 49  86  69      a(v(1:3)) = b(2:4)                      min 49(1)  max 221(0)
0   0   0   0
0   1   0   0       fail = .false.                          min 0(3)   max 1(1)
31  31  46  26      do i = 1, n                             min 26(3)  max 46(2)
0   0   0   0         if (i .lt. n-2) then
190 189 134 129         if (a(i) .ne. -1) fail = .true.     min 129(3) max 190(0)
0   0   0   0         else
66  55  55  42          if (a(i) .ne. n+2-i) fail = .true.  min 42(3)  max 66(0)
0   0   0   0         endif
0   0   0   0       enddo
0   0   0   0       if (fail) then
0   0   0   0         print *, 'fail: "a(v(1:3)) = b(2:4)"'
0   0   0   0       else
6   7   8   2         print *, 'pass'                       min 2(3)   max 8(2)
0   0   0   0       endif
0   0   0   0
0   0   0   0    ccccc
70  65  46  40      a = -1                                  min 40(3)  max 70(0)
16  11  10  9       a(v(2)) = b(2)                          min 9(3)   max 16(0)
0   0   0   0
0   0   0   0       fail = .false.
94  32  59  35      do i = 1, n                             min 32(1)  max 94(0)
0   0   0   0         if (i .eq. n-1) then
24  11  14  13          if (a(i) .ne. 2) fail = .true.      min 11(1)  max 24(0)
0   0   0   0         else
249 219 202 217         if (a(i) .ne. -1) fail = .true.     min 202(2) max 249(0)
0   0   0   0         endif
0   0   0   0       enddo
0   0   0   0       if (fail) then
0   0   0   0         print *, 'fail: "a(v(2)) = b(2)"'
0   0   0   0       else
7   3   11  1         print *, 'pass'                       min 1(3)   max 11(2)
0   0   0   0       endif
0   0   0   0
```

Figure 1: Detailed view (column a), with additional windows to show source code (column b), and sorted performance data per source line (column c).



```
56566: ldah   v0,-2(gp)
56567: ldl    v0,23864(v0)
56568: lda    t0,-1(zero)
56569: ldah   a0,-2(gp)
56570: lda    a0,23864(a0)
56571: cmovlbs   v0,t0,s1
56572: lda    a1,-14936(gp)
56573: lda    a2,-14928(gp)
56574: ldq    t12,-32488(gp)
56575: bis    zero,zero,zero
56576: bis    zero,zero,zero
56577: bis    zero,zero,zero
56578: blbc   s1,0x12000f130
56579: bis    zero,0x16,s4
56580: bis    zero,zero,zero
56581: ldah   t0,-8194(gp)
56582: bis    zero,0x6,a1
56583: bis    zero,zero,zero
56584: lda    a4,64(s6)
56585: bis    zero,zero,zero
```

Figure 2: Assembler instructions of a selected area.

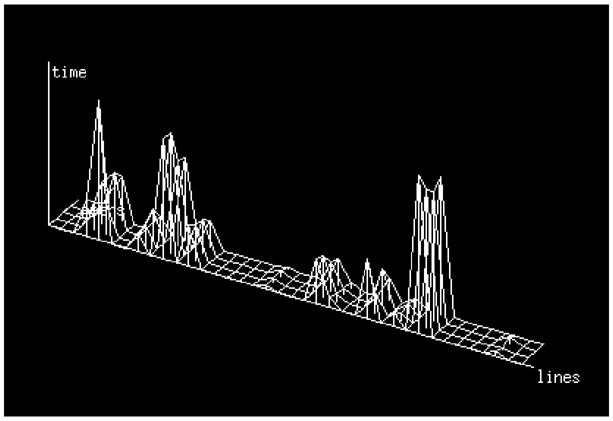


Figure 3: Three-dimensional performance profile.

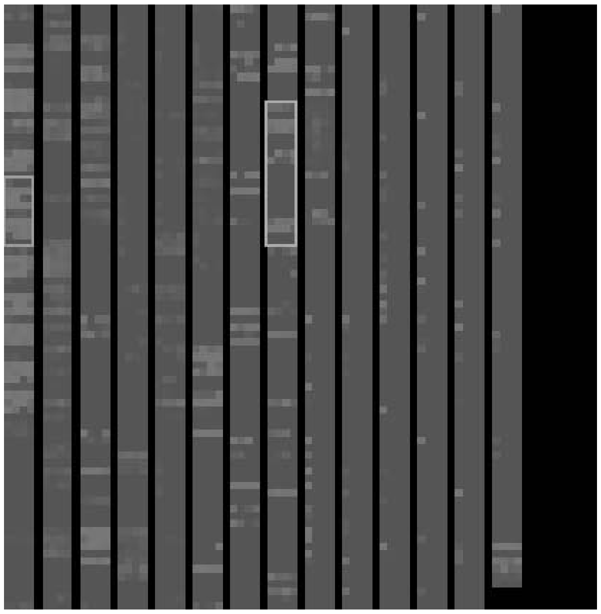
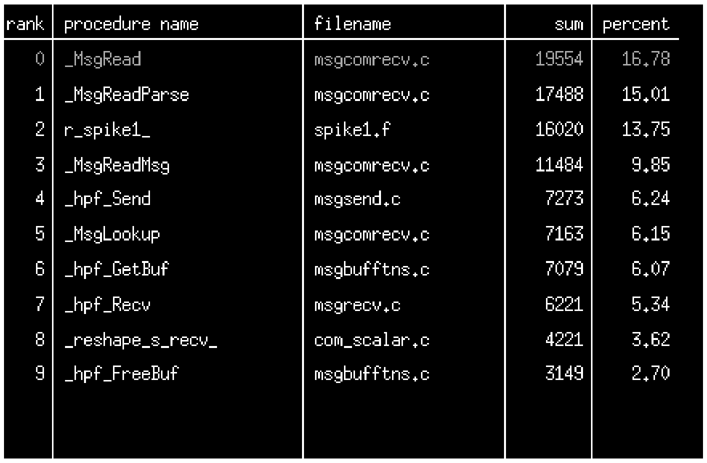


Figure 4: Overview window.



rank	procedure name	filename	sum	percent
0	_MsgRead	msgcomrecv.c	19554	16.78
1	_MsgReadParse	msgcomrecv.c	17488	15.01
2	r_spike1_	spike1.f	16020	13.75
3	_MsgReadMsg	msgcomrecv.c	11484	9.85
4	_hpf_Send	msgsend.c	7273	6.24
5	_MsgLookup	msgcomrecv.c	7163	6.15
6	_hpf_GetBuf	msgbufftns.c	7079	6.07
7	_hpf_Recv	msgrecv.c	6221	5.34
8	_reshape_s_recv_	com_scalar.c	4221	3.62
9	_hpf_FreeBuf	msgbufftns.c	3149	2.70

Figure 5: Ranked procedure window.

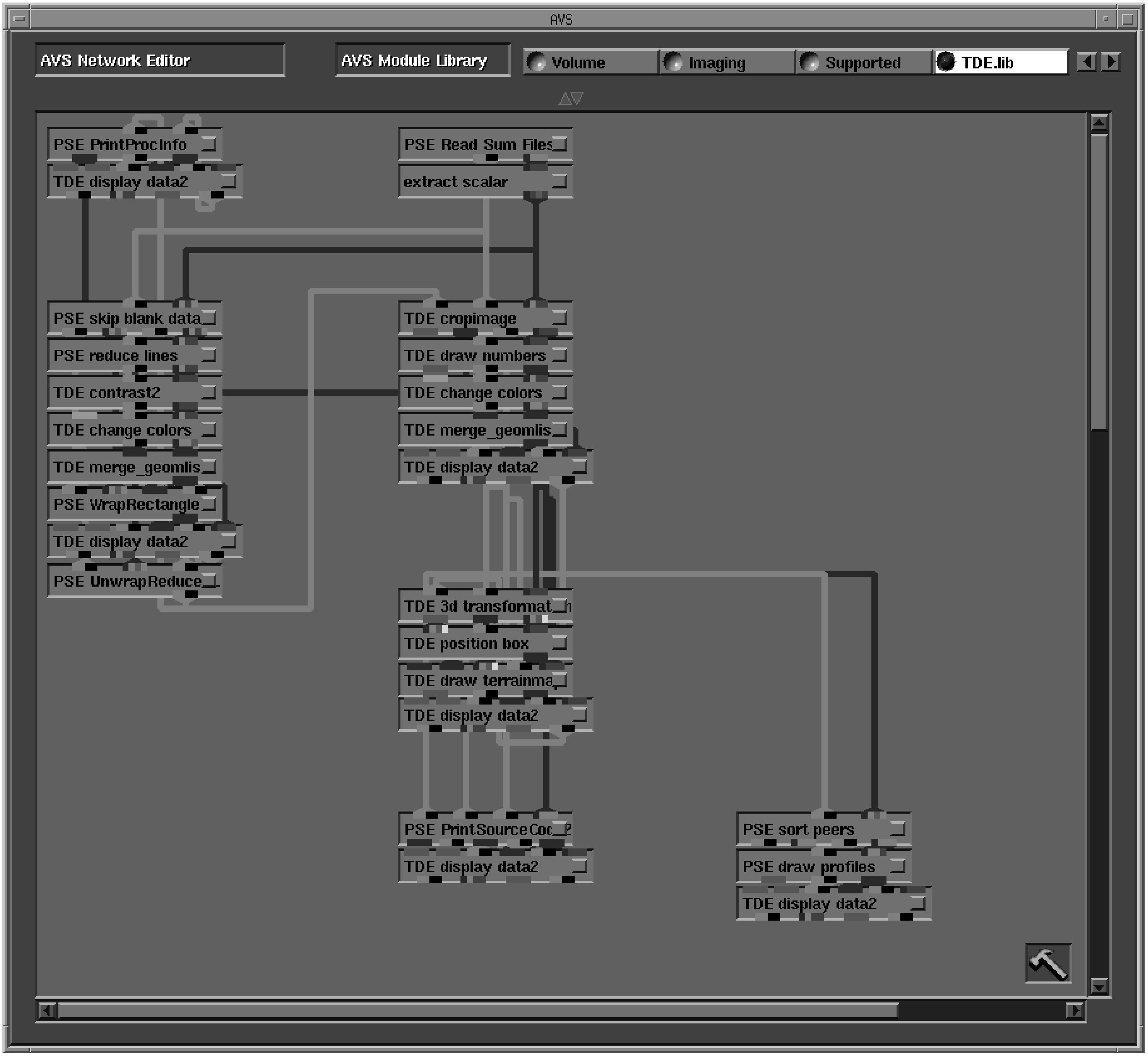


Figure 6: TDE/AVS data flow network of the performance profiler.

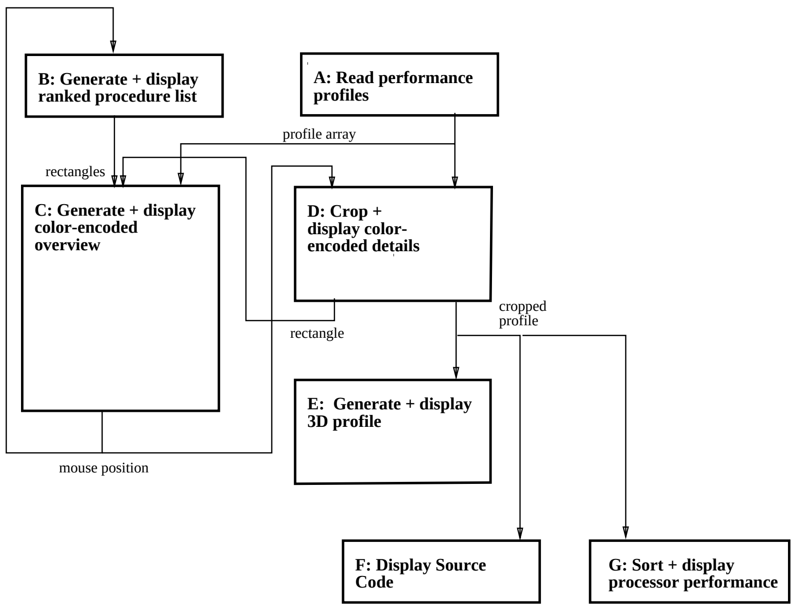

B: Generate + display ranked procedure list
A: Read performance profiles
profile array
rectangles
C: Generate + display color-encoded overview
D: Crop + display color-encoded details
cropped profile
rectangle
E: Generate + display 3D profile
mouse position
F: Display Source Code
G: Sort + display processor performance

Figure 7: Schematic overview.

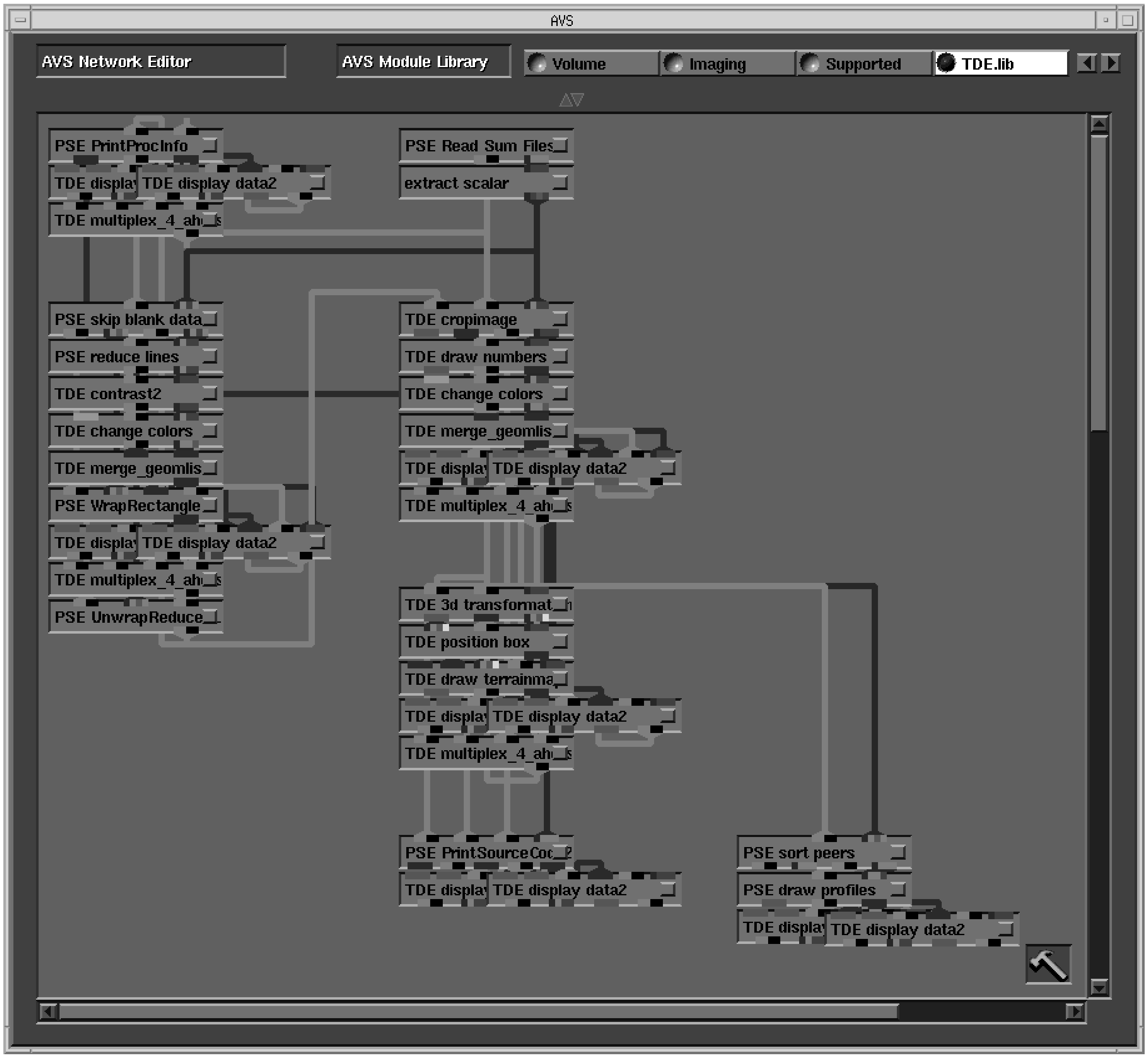


Figure 8: TDE/AVS data flow network for telecollaborative performance visualization.