

Concept Visualizations of Computer Programs

Brian J. d'Auriol

Department of Computer Science
The University of Texas at El Paso
El Paso, TX, USA 79968
Email: dauriol@acm.org

Abstract—A major issue facing the programming world today is the quick and efficient understanding of existing program code by programmers and software engineers. Visualization of concepts inherent in the program code is proposed as a new mechanism to facilitate program comprehension. The premise of this research is that program comprehension is primarily based on the reader's conceptual formation of program code fragments. This paper identifies two novel visualization models called the Program-Scientific and the Conceptual Crown Visualization models; and presents preliminary visualization studies based on understanding programs from data processing, parallel computing and high performance computing applications.

I. INTRODUCTION

A major issue facing the programming world today is the quick and efficient understanding of existing program code by programmers and software engineers. The ability to easily comprehend program code results in time and financial savings in such situations as multiple-person coding teams with a high employee turn-around. Code maintenance can also be facilitated when programmers better understand the complexities inherent in the existing code. Today's computing infrastructure is significantly composed of distributed applications developed in and executed under such infrastructure technologies as Microsoft's .NET and CORBA. Client-server and the emerging Peer-to-Peer computing models further support distributed applications. Methods and techniques aimed at facilitating programmer and software engineer understanding of these types of programs should facilitate better code development and maintenance. In addition, high performance computing can also benefit from better understanding by programmers, especially, in interpreting legacy codes.

Program understanding is commonly facilitated by good source code layout, good data and procedural abstractions and good internal and external documentation. Many common techniques such as pretty-printing, indentation and the use of extra blank lines provide visual clues about the meaning and purpose of code fragments. Data and procedural abstractions in the code are captured by data structures and the subdivision of operations into tasks, objects and procedures. Other techniques such as commenting, appropriate identifier naming and external documentation provide heightened information about the concepts inherent in the program.

Despite the long history of writing readable code, understanding programs remains a daunting task: large code size, many supported functions and, especially, the additional

complexities found in distributed and web-based computing all contribute to difficulties in understanding programs.

The premise of this research is that program comprehension is primarily based on the reader's conceptual formation of program code fragments. The reader considers input from the visual display, functional abstractions and documentation of the code. Connections between these inputs and concepts are then formed. However, there are several impeding factors: the visual display and documentation may be incorrect, the volume of available information requires time to assimilate and the code itself may be hard to read. Hence, program comprehension requires the formation of concepts about the purpose and intended actions of programs. However, present day mechanisms do not facilitate this process.

Visualization of concepts inherent in the program code is proposed as a new mechanism to facilitate program comprehension. The idea behind program concept visualizations is that there must have been a reason why a programmer wrote a particular code fragment; it should be possible to determine that reason by examining the code itself, perhaps, supplemented with documentation.

Models to support the visualization of concepts in programs are described and applied in this paper. The emphasis is to consider distributed, high performance and data intensive programs.

This paper is organized as follows. Section II describes background information, in particular, concepts and visualization. Section III gives an overview of the two visualization models as well as the visualization framework used in later sections of this paper. Section IV presents results of the applications of these visualization models in several case studies. Conclusions are given in Section V.

II. CONCEPTUALIZATION AND VISUALIZATION

This section describes concept identification and representation as well as conceptual and program visualization.

Concept identification and representation has been applied in different fields. Gärdenfors in [1] advances the model of Conceptual Spaces. The model consists of the two components: *Quality Dimensions* and *Domains*. A quality dimension is used to represent an attribute or property of the modeled entity. A domain is a set of integral quality dimensions that are separable from all other dimensions. An application of conceptual spaces for robotics is discussed in [2]; an application for intrusion detection in computer security is presented

in [3]. Schütz in [4] is cited in the context of Latent Semantic Analysis. Schütz presents a model for sense discrimination, that is, the identification of meaning to ambiguous words in the context of information retrieval in large text databases. A cluster-based approach in a vector space formed by conceptual dimensions forms the basis for the conceptual representation. Geometric Representation of Programs (GRP), proposed by d’Auriol [5], [6], uses computational awareness dimensions to represent conceptual information about computer programs. Computations, data or conceptual properties are mapped to coordinates in a metric space and grouped within polytope-type objects. Relationships between the entities or between the groups are represented by graphs. The polytopes form abstractions about the program code. These models, although from three different fields, have several similarities: a dimensional space is used to represent conceptual information together with techniques to identify and represent information in such a space. Other conceptual models include Formal Concept Analysis [7] which is a mathematical formalization for concepts.

Some work on conceptual analysis of programs is discussed in [8]. The authors propose the definition: “Concepts are units of human knowledge that can be processed by the human mind ... in one instance”. The authors use this definition in work that facilitates program comprehension. A number of case studies are also discussed.

There has been a lot of work in program visualization to facilitate the learning of algorithms, see for example, the discussions in [9]. Visualization tools for program understanding is the subject of [10]. A conceptual visualization is described in [11] where a “... viewer may follow the [concept] map which branches off to various nodal points. Each of these nodal points represents a single theory.” Radial visualization also referred to as RadViz is described in [12], [13]. This visualization technique places descriptive words that could represent concepts around a circle. Information items have corresponding weighted vectors, the length of which corresponds to the number of words. Information items are plotted within the circle according to these weight vectors. In this way, the information is displayed relative to the combination of contained words or concepts. Other research areas that combine visualization and program comprehension include software engineering. The use of visualization for program comprehension, indeed, has seen broad research activity, and a more detailed discussion is beyond the scope of this paper. Two observations are: (a) there is general consensus that program visualization can facilitate program comprehension, and (b) there remain open questions.

The research reported on in this paper deals with similar issues. However, A different and somewhat more formalized definition of *concept* motivates the proposed work.

III. VISUALIZATION OVERVIEW

The visualization approach is to construct a hierarchy of relationships among the program statements. Low-level semantics are bound with the individual statements, thereby,

establishing a semantic interpretation of each statement. The hierarchy itself consists of higher-order relations, i.e., relations between (lower level) relations. This allows for the propagation of the semantic interpretations upwards through the hierarchy. The semantic propagation results in more and more abstract descriptions of the code fragments. The hierarchy is organized in terms of levels of relations. A relation’s level in the hierarchy describes the relative abstractness of that relation. Concepts are identified in terms of the highest-order relations.

There are two types of relations. Semantic relations bind semantics to individual program statements whereas constructive relations extend the semantics by creating more abstract descriptions that apply to the group of participating relations. In this work, only binary relations are considered. Constructive relations are restricted such that one of the two participating relations must be at a preceding level of abstraction. This ensures a consistent upwards propagation of the semantics descriptions. Let $R^1 = (S, d)$ denote a Level 1 relation that binds the semantics d to its associated statement S . Higher level relations are denoted by $R^l(x, y, d)$ where l represents the relation’s level, and x and y are lower level relations, with either x or y subject to restriction that it is at the preceding level. Subscripts denote specific relations.

The proposed concept visualization system is based on the Advanced Relation Model for Program Visualization (ARM 4 PV) [14]. The system is two phased. Program code is analyzed so as to determine the relation hierarchy in the first phase. In the second, the relations are visualized by one or more visualization models. Two visualization models are described in this paper: the Conceptual Crown Visualization (CCV) model [15], and the Program-Scientific Visualization (PSV) model [16].

The CCV model provides concept visualizations to facilitate the viewer’s better understanding of the concepts inherent in the programming. There are two basic visualizations that are defined: a line structure and a space structure visualization. The former renders selected relations in the relation hierarchy as either single vertical lines (Level 1) or dual piece-wise single point-connected lines (higher levels) whereas, the latter renders concepts as a convex hull of the participating lower-level relations. Relations are graphed in the $x - y$ plane. The linear x -axis is continuous. In this paper, Level 1 relations are represented as vertical lines, $(\overline{S_i, 0}), (\overline{S_i, 1})$. This means that program statements are mapped to integer coordinates in their lexicographic order. A higher level relation, $R^l = (R^i, R^j, d)$ for $l > 1$, is represented as: $(x_1, i)(x_2, l), (x_2, l)(x_3, j)$ where (x_1, i) and (x_3, j) are points for relations R^i and R^j , respectively; and, x_2 is the midpoint of the minimum and maximum extents of all the participating Level 1 relations in the R^l . For perceptive reasons, the x -axis is mapped to a circle thereby creating a cylinder shaped visual object. This enables: (a) immersive visualization by allowing the viewpoint to be placed in the center of the circle, (b) a zooming operation by providing greater forefront focus while compressing the information in the peripheral area, and (c) greater information

density by providing up to twice the amount of information displayed on the screen.

The PSV model uses traditional scientific visualization techniques like contouring to provide insight into program related information. The application of the PSV model in this paper follows from the discussions in [16]. The three attributes of computation, communication and input/output are represented as a three element vector, each element defined on zero through one inclusive. Each relation in the hierarchy is associated with a particular instance of this attribute vector. Averaging between corresponding elements is used to propagate attribute values upwards through the hierarchy:

Given

$$\begin{aligned} R_1^1 &= (S_1, d_1, (e_{1,1}, e_{1,2}, e_{1,3})) \\ R_2^1 &= (S_2, d_2, (e_{2,1}, e_{2,2}, e_{2,3})) \end{aligned}$$

then

$$R_1^2 = (R_1^1, R_2^1, d_\alpha, w) \quad (1)$$

where

$$w = \left(\frac{(e_{1,1} + e_{2,1})}{2}, \frac{(e_{1,2} + e_{2,2})}{2}, \frac{(e_{1,3} + e_{2,3})}{2} \right) \quad (2)$$

where d_α represents the semantic abstraction of d_1 and d_2 . Although the PSV model allows for many scientific visualization techniques, the one selected in this paper is a 3-D plot with the axes corresponding to computation, communication and input/output. Glyphs are used to highlight the requirement values, both color and size represent the relation's level. This visualization provides for the requirement to identify the computational, communication and input/output components in a given program.

IV. APPLICATIONS

This section presents preliminary visualization studies based on understanding programs from data processing, parallel computing and high performance computing applications. Visualizations are presented and discussed. The emphasis in this study is on the reasonable application of the visualization models. Perceptive studies are not included in this paper.

A. Swap Program

Figure 2 and Figure 3 show Conceptual Crown line structure and space structure, respectively, concept visualizations of the swap program in Figure 1. The swap program consists of two procedures, main and swap, main interactively reads input and calls swap to exchange the values. This is a very simple program; its purpose in this discussion is to illustrate principals of conceptual visualization. The following observations from the visualizations are made.

- The most abstract description of this code, that is, the swapping (exchange) of the user input values, is indicated by the green highlighted Level 4 relation in Figure 2

and by the wide Level 4 relation (triangle) at the back of Figure 3. This relation connects to Level 3 relations in separate relation sub-sets (Figure 2). These sub-sets themselves form concepts of, respectively left-to-right, the swapping function (statements 1 through 6) and the main function (Statements 7 though 16). The middle section of Figure 3 shows these Level 3 concepts.

- The concept of the swap function (Statements 1–6) are not well formulated for two reasons, first, the Level 4 relation highlighted in cyan in Figure 2 is labeled ‘print swapped values’, yet, it is the Level 3 relation that is connected outside of the function, second, there is another Level 3 relation in the main function that also prints the values (shown highlighted in red), therefore, there is operational duplication. The visualization suggests that this function is poorly written and the code should be reviewed, particularly with respect to its coupling to calling functions.
- The concept of the main function (Statements 7–16) are not well formulated since there are three Level 3 relations at the highest level in this concept, but only one of which participate in the primary Level 4 relation (Figure 2). The visualization suggests either that this function may be poorly written or that the relational hierarchy does not sufficiently describe the full set of relations and concepts associated with this function. In this case, there is a lack of supporting relations, particularly at the higher levels, which contribute to the vagueness of the concept.
- Figure 3 shows in landscape mode, the Level 2, 3 and 4 concepts where, at the second and third level, the concepts are distinctly separable in terms of the two procedures.

Overall, this visualization example illustrates concept identification, separation and assessment. The primary concept of swapping user input values is evident from the visualization. The two sub-concepts corresponding to the two functions are also readily identifiable. The separation of these concepts is also evident from the visualization. In terms of assessment, several observations are made. The visualization indicates that the main purpose of the program appears to be clearly structured and supported by the program code. The visualization shows high cohesion and low coupling between the functions, thereby suggesting a well developed program. However, the purposes of each of the two functions appear not to be as well structured nor well supported by the program statements. Furthermore, the visualization suggests that a programmer should review the code in each of the two functions, for example, to determine the reason for the duplicate value output.

B. Parallel Computing

The study focuses on the introductory textbook example of the MPI parallel program in Figure 4 to calculate the value of π using numerical integration [17]. The Message Passing Interface standard (MPI) provides for a common library implementation of message-based communication amongst parallel

```

1 #include <stdio.h>
2 void swap(float *x, float *y)
3 {
4     float t;
5     t = *x;
6     *x = *y;
7     *y = t;
8     printf("Values in swap %f,%f\n", *x, *y);
9 }
10 main()
11 {
12     float x, y;
13     printf("Please input 1st value: ");
14     scanf("%f", &x);
15     printf("Please input 2nd value: ");
16     scanf("%f", &y);
17     printf("Values BEFORE swap %f,%f\n", x, y);
18     swap(&x, &y);
19     printf("Values AFTER swap %f,%f\n", x, y);
20     return 0;
21 }

```

Fig. 1. Swap program example.

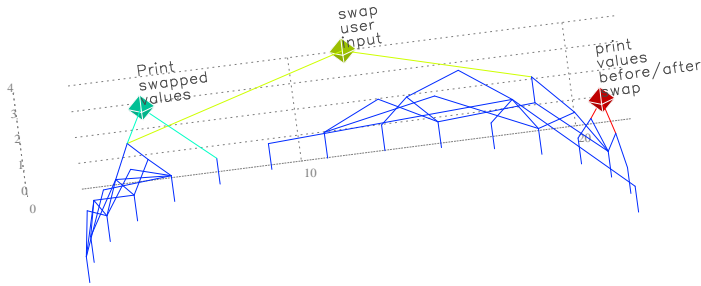


Fig. 2. CCV visualization of swap program: 180°, radius 10 line structured visualization with glyph and color highlights.

```

1 #include "mpi.h"
2 #include <stdio.h>
3 #include <math.h>
4
5 double f( double );
6
7 double f( double a )
8 {
9     return (4.0 / (1.0 + a*a));
10 }
11
12 int main( int argc, char *argv[] )
13 {
14     int done = 0, n, rank, numprocs, i;
15     double PI25DT = 3.141592653589793238462643;
16     double mypi, pi, h, sum, x;
17     double startwtime = 0.0, endwtime;
18     int namelen;
19     char processor_name[MPI_MAX_PROCESSOR_NAME];
20
21     MPI_Init(&argc,&argv);
22     MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
23     MPI_Comm_rank(MPI_COMM_WORLD,&rank);
24     MPI_Get_processor_name(processor_name,&namelen);
25
26     fprintf(stderr,"Process %d on %s\n", rank, processor_name);
27
28     n = 0;
29     while (!done) {
30         if (rank == 0) {
31             printf("Enter the number of intervals: (0 quits) ");
32             scanf("%d",&n);
33             startwtime = MPI_Wtime();
34         }
35         MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
36         if (n == 0)
37             done = 1;
38         else {
39             h = 1.0 / (double) n;
40             sum = 0.0;
41             for (i = rank + 1; i <= n; i += numprocs) {
42                 x = h * ((double)i - 0.5);
43                 sum += f(x);
44             }
45             mypi = h * sum;
46
47             MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
48                     MPI_COMM_WORLD);
49
50             if (rank== 0) {
51                 printf("pi is approximately %.16f, Error is %.16f\n",
52                       pi, fabs(pi - PI25DT));
53                 endwtime = MPI_Wtime();
54                 printf("wall clock time = %f\n",
55                       endwtime-startwtime);
56             }
57         }
58     }
59     MPI_Finalize();
60     return 0;
61 }

```

Fig. 4.

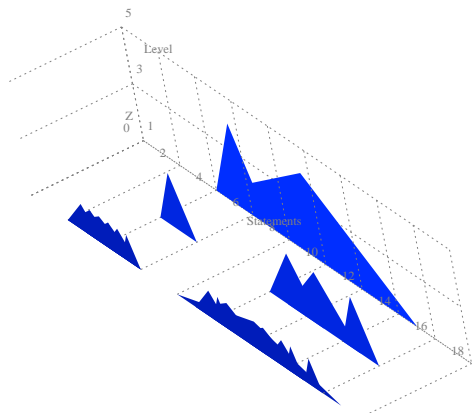


Fig. 3. CCV visualization of swap program: landscape space structured visualization.

processors. MPI is highly portable which means that MPI code can be compiled and executed on many widely variant parallel systems. A fundamental concept in such parallel programs is that of the granularity of the parallel code, that is, the amount of computation to communication. This impacts both the performance of the code as well as how the computation and communication is specified in the code. The latter is of interest in this visualization study. Specifically, the purpose of this study is to both quantify and qualify the concepts relating to parallel program specification, including, granularity. First, the PSV model is applied to visualize the quantification of the computation, communication and input/output aspects; then the CCV model is applied to qualitative visualize the parallel computing and program structure concepts.

Figures 5, 6 and 7 show the application of the PSV model as 3-D plot of the computation, communication and input/output attributes. The axes in these figures are labeled, respectively, "Cp", "Cm" and "I/O". The space forms a unit cube with the

origin at $(0, 0, 0)$. The level of the relations are indicated both by color and size, a color legend indicates the color mapping, and, the larger the size, the higher the level. Three isosurfaces on the back-planes are used to show linear color interpolation throughout the unit cube space, and hence, indicate the overall dominance of the attributes with respect to the program code. Figure 5 is displayed with a slight forward tilt, the “Cm” axis is vertical in this figure. Figure 6 is an orthogonal view that includes depth cues and shows input/output versus computations, while, Figure 7 is an orthogonal view, also with depth cues, that shows communications versus computations.

Collectively, these figures are interpreted as follows.

- There are significant amounts of input/output and computations, but much less communications.
- Communication is supported by two lower level concepts, furthermore, combining with the previous observations, it is deduced that the communication relations participate in few higher level concepts.
- The Level 3 concept with its computation component of zero, shown in green in the middle of the isosurface plane at the back-left side of Figure 5, suggests a communication of an input/output value.
- The Level 4 concept, shown in yellow, has no communication component, hence, an important conceptual characterization of this program is its computation based on input/output.
- The Level 5 concept, shown in red, has a communication component, moreover, it indicates, roughly, a 40% computation component, a 50% communication component and a 20% input/output component.

Figures 8, 9, 10 and 11 show line structured CCV conceptual visualizations. Color is used to identify specific higher level relations together with the respective participating lower relations. Semantic annotations appear for most of the Level 3 relations as well as for the Level 5 relation. Figure 8 shows a non-glyph highlighted image whereas the other three show glyph highlighted images. Figure 9 is a zoomed-in image where the glyphs highlight the concepts that are of interest. The glyphs in Figure 10 highlight the relations that participate in the formation of the concept relations (note, the glyph coloring may not show all of the relations that it participates in). Lastly, Figure 11 illustrates a rotated image; note that all of the information is still displayed and hence, global context of information is maintained. Yet, the viewer can focus in on the beginning parts of the program.

Collectively, the CCV visualizations are interpreted as follows.

- There are two distinct code fragments, the first includes Statements 7 through 9, the second, Statements 12 through 59. These fragments correspond to the functions `f` and `main`. These are shown with clear separation in the visualization.
- The structure of the main function is clearly visible in the visualizations. The ‘fan’ type patterns in the Level 2 relations indicate the scope of the various constructs

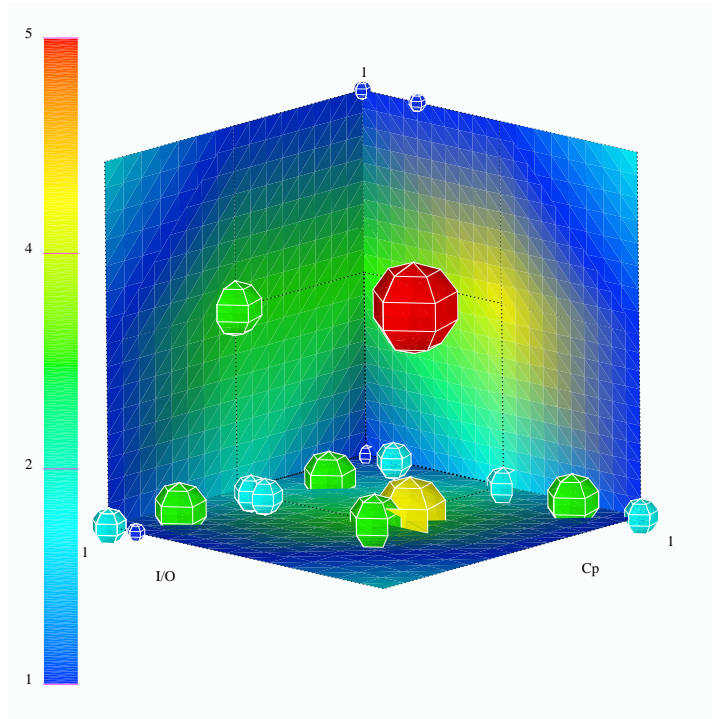


Fig. 5.

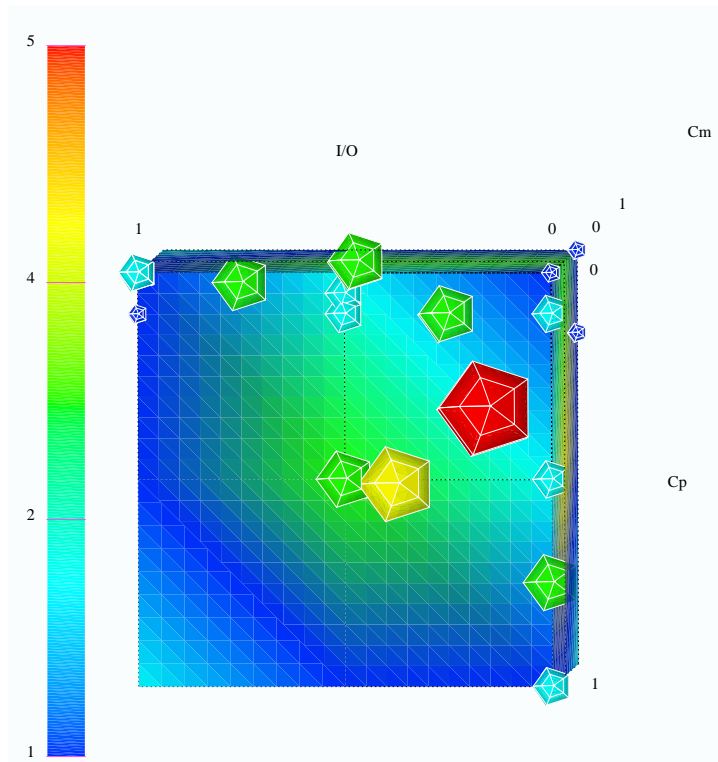


Fig. 6.

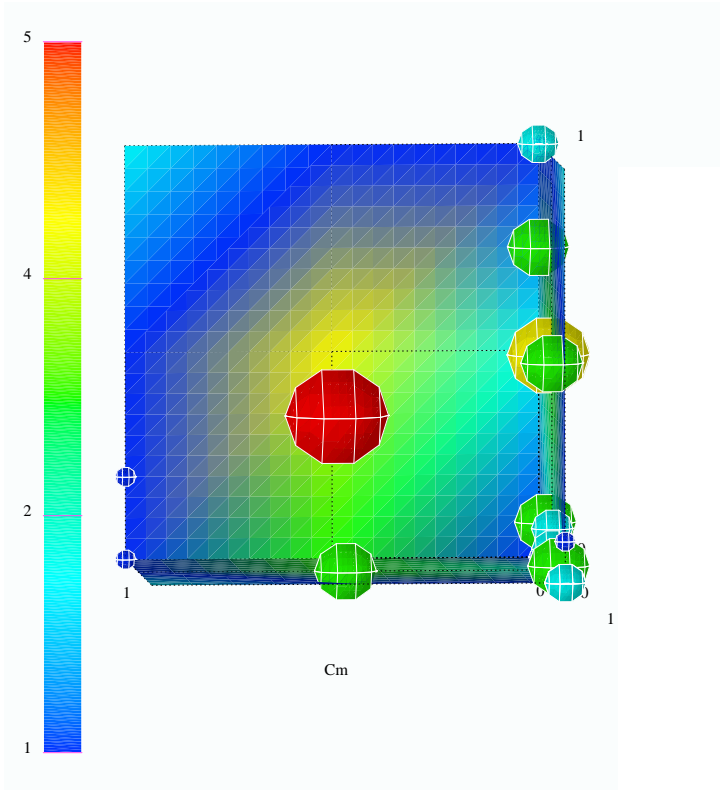


Fig. 7.

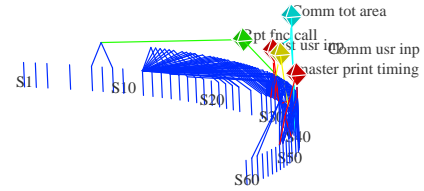
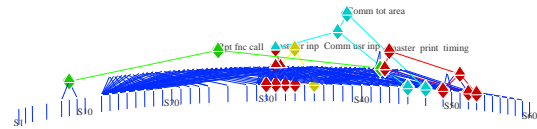


Fig. 11.

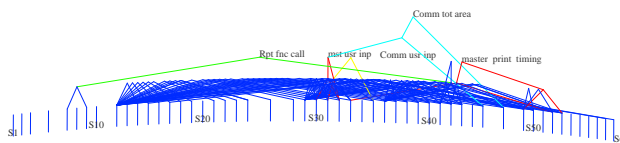


Fig. 8.

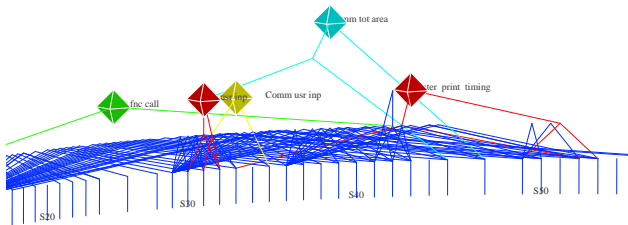


Fig. 9.

that make up the main function. The `while` construct extends from Statement 29 through 53; an `if` construct extends from Statement 29 through 33; a second `if` construct extends from Statement 36 through 53; the inner-most `for` construct extends from Statement 41 through 43; and, the inner-most `if` construct extends from Statement 49 through 53. These patterns are easily discernible in the zoomed image, Figure 9.

- The two red-identified Level 3 concepts pertain to the master/slave concept of parallel programming, prevalent in MPI codes. Respectively, the two concepts are “interactive user input” and “output timings”. The latter is composed of two Level 2 relations, the righter-most pertains to master output. From combining the structure observations from the previous observation with the statements participating in these master concepts, it is observed that the master concepts are governed by conditional constructs: this programming style is also prevalent in MPI codes. Since lexicographic statement ordering is displayed left-to-right in the visualizations, it is further observed that the code segments in-between the master concepts are not within the master-only code, hence, it is deduced that all run-time instances of this code perform the computations indicated by the in-between relations. Lastly, there is a weak connection between the two master concepts as shown by a single Level 2 relation. This, exactly, pertains to the timing of the MPI code, the start timer is located in the first master concept, the end timer in the second concept. However, the visualization only weakly suggests this by the concept notated as “master print timing”.
- Following from the preceding observations, the function `f` is called repeatedly in a `for` construct in each of

the run-time instances. This is shown by the highlighted green concept.

- The Level 3 yellow highlighted relation represents the concept of communicating the user input. It is suggested by its participating relations that the user input from the master is communicated in a code fragment that is not part of the master. In other words, the communication statement, Statement 35 (identified by the right-most yellow highlighted line), occurs in each run-time instance. Since there is no appearance of a matching read-write, it is deduced that this is a broadcast collective communication.
- The Level 5 concept, “Comm tot area”, shown in cyan, is based on a Level 4 concept “Computer Aggregate area”. This suggests that the computation computes an area value which is then communicated. The communication is performed by Statement 47 as indicated by the right-most cyan line between the Level 5 concept and the Level 1 relation. Since there does not appear to be any further computation occurring after the communication, it is deduced that the communication is an aggregation, possibly, of the form of a reduction during the communication. This deduction is confirmed by observing that Statement 47 in Figure 4 is an `MPI_Reduce`.

These observations from the visualization show that the concepts of parallel programming, including identification of computational, communication and user input/output components can be extracted from the visualization. In this regard, the CCV model visualization appears successful in the *qualification* of the parallel program.

C. High Performance Computing

This section reports on work-in-progress regarding the application of the visualizations to a FORTRAN code. The program consists of recurrence equations that calculate the mean gain and the variance of the gain in APDs (Avalanche Photo Diode) with the inclusion of the dead-space effect. This study is in its infancy, however, is included here to illustrate the application of the visualizations in realistic situations. Figure 12 shows a Level 1 with a few scattered Level 2 and 3 concepts. This visualization is mapped onto a three-quarter circle. Figure 13 is a zoomed-in image. The visualization indicates three things: a) there are several hundred program statements in the program, b) there are low-level concept connections between the code at the beginning of the program, and code near the end of the program (this corresponds with COMMON block access), and c) that the set of concepts highlighted in Figure 13 form a tightly-coupled unit (this corresponds with a subroutine).

One aspect of parallelization is to consider the data dependencies between statements, in particular, when evaluating which segments of memory can be accessed independently. The COMMON block identified in this visualization means increased dependences between subroutines which may reduce the potential parallelizations of this program. This study successfully shows that the visualizations could assist in

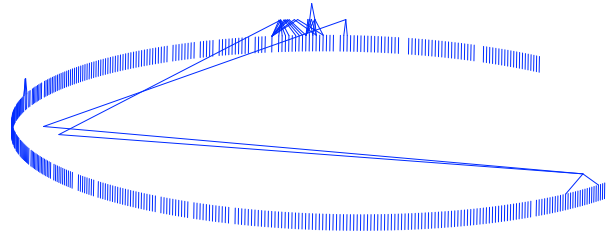


Fig. 12.

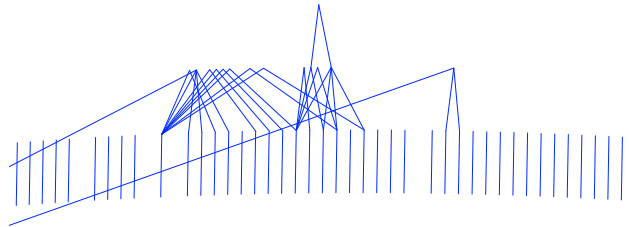


Fig. 13.

understanding the concepts related to data dependency analysis and parallelization. However, clearly, an extensive relational set describing the program is required before any further useful visualization analysis can be performed.

V. CONCLUSIONS

This paper describes two visualization models that may facilitate program comprehension. The visualization models, called the Program Scientific Visualization model (PSV) and the Conceptual Crown Visualization model (CCV), can be applied to general program code. The emphasis on this paper, and that which motivates this work, is the identification of conceptual level information relating to computation and communication in parallel and distributed programs. Program comprehension, program visualization, conceptual representation and visualization as well as software engineering are all fields that relate with the proposed models.

The PSV and CCV models are applied to three different programs, and analyses are conducted. The results strongly suggest that these visualizations can provide useful information regarding: a) concept identification and separation, and b) specific to the motivating interests, computation and communication related concepts. Further details about one of these applications appear in [18].

The visualizations are based on information extracted from a static analysis of a given program and which is represented in a relational form. To the extent that such a relational database is complete and consistent, this paper suggests the usefulness of the visualizations. However, there are several

current limitations regarding the construction of the relational database, for one, the procedure is not yet automated, therefore, the information used in this paper was based on hand-crafted analysis; and second, a semantic engine is required to formulate and propagate the concepts upwards in the relational hierarchy. In the future, these issues need to be addressed. Also, subject tests need to be conducted to explore pre-attentive information processing and manipulation available with these visualizations, as well as the extent of required analytic processing. Additional visual elements, for example color sequences and interactive visualizations, should also be explored. Lastly, the current visualization prototypes need to be enhanced, for example, to provide convex hull identification and rendering in space structured CCV visualizations.

REFERENCES

- [1] P. Gärdenfors, *Conceptual Spaces, The Geometry of Thought*. Cambridge, MA: MIT Press, 2000.
- [2] A. Chella, S. Gaglio, and R. Pirrone, "Conceptual representations of actions for autonomous robots," *Robotics and Autonomous Systems*, vol. 34, pp. 251–263, 2001.
- [3] A. Akinsanmi, "A conceptual space model for intrusion detection," Master's thesis, Department of Computer Science, The University of Texas at El Paso, December 2002.
- [4] H. Schütz, "Automatic word sense discrimination," *Computational Linguistics*, vol. 24, no. 1, pp. 97–124, 1998.
- [5] B. J. d'Auriol, "Expressing parallel programs using geometric representation: Case studies," in *Proc. of the IASTED International Conference Parallel and Distributed Computing and Systems (PDCS'99)*, Cambridge, MA, USA, Nov. 1999, pp. 985–990.
- [6] —, "A geometric semantics for program representation in the polytope model," in *Proc. of the Twelfth International Workshop on Languages and Compilers for Parallel Computing LCPC'99, Aug. 4-6, The University of California, San Diego, La Jolla CA USA*, ser. Lecture Notes in Computer Science, Vol. 1863. Springer-Verlag, 1999, pp. 451–454.
- [7] B. Ganter and R. Wille, *Formal Concept Analysis Mathematical Foundations*. Springer, 1999.
- [8] V. Rajlich and N. Wilde, "The role of concepts in program comprehension," in *Proc. of the 10th International Workshop on Program Comprehension*, June 2002, pp. 285–288.
- [9] J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, Eds., *Software Visualization, Programming as a Multimedia Experience*. The MIT Press, 1998.
- [10] S. Tilley and H. Shihong, "On selecting software visualization tools for program understanding in an industrial context," in *Proc. of the 10th International Workshop on Program Comprehension*, June 2002, pp. 285–288.
- [11] R. Merritt, "Intentions: Logical and subversive. the art of marcel duchamp, concept visualization, and immersive experience," in *Proc. of the Fifth International Conference on Information Visualisation*, London, UK, July 2001, pp. 233–240.
- [12] P. Hoffman, G. Grinstein, K. Mark, I. Grosse, and E. Stanley, "DNA visual and analytic data mining," in *Proceedings of the 23rd International Visualization '97*. Phoenix, AZ, USA: IEEE, Oct. 1997, pp. 537–441.
- [13] P. Au, M. Carey, S. Sewraz, Y. Guo, and S. Ruger, "New paradigms in information visualization," in *Proceedings of the 23rd International ACM SIGIR Conference*. Athens, Greece: ACM Press, July 2000, pp. 307–309.
- [14] B. J. d'Auriol, "Advanced relation model for program visualization (ARM 4 PV)," in *Proc. of The 2004 International Conference on Modeling, Simulation and Visualization Methods (MSV'04)*, Monte Carlo Resort, Las Vegas, Nevada, USA., June 2004, (to appear, post-conference proceedings).
- [15] A. Gajjala, "A model for visualization of program conceptual information," Master's thesis, Department of Computer Science, The University of Texas at El Paso, May 2004.
- [16] R. Policherla, "Scientific visualization techniques for program visualization," Master's thesis, Department of Computer Science, The University of Texas at El Paso, May 2004.
- [17] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI, Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, 1999.
- [18] B. J. d'Auriol, "A concept visualization study of a parallel computing program," in *Proc. of the 6th International Workshop on High Performance Scientific and Engineering Computing with Applications (HPSEC-04) in conjunction with the International Conference on Parallel Processing (ICPP-2004)*, Montreal, Quebec, Canada, Aug. 2004, (To appear).