A Concept Visualization Study of a Parallel Computing Program

Brian J. d'Auriol
Department of Computer Science
The University of Texas at El Paso
El Paso, TX, 79968, USA
dauriol@acm.org

Abstract

Conceptual visualization is proposed in this paper to facilitate reading and understanding parallel programs. This addresses the need to better understand high performance program codes, especially, in an environment where the codes reflect the diversity in architecture as well as in programming and language models. A relational model is developed to represent abstractions about the program. High level abstractions are termed concepts in this work. A visualization technique based on the relation hierarchy and composed of two distinct visualization models is further proposed in this paper. A study of a simple MPI parallel program is made. Various visualizations from both of the proposed visualization models are included and discussed. Combined, the visualizations provide for qualitative and quantitative aspects relating to the concepts in the given program. This study suggests that the proposed visualizations do facilitate the understanding of parallel programs.

1 Introduction

High performance computing typically involves solving computational or data intensive problems on high performance computers. Programs are typically written in FORTRAN or C, however other languages can been used. Often, these programs are vectorized or parallelized for performance. High performance computers include SIMD and MIMD variations of shared and distributed memory computers with optimized scalar processing capability together with a high-speed, high-bandwidth and optimized memory and processor interconnect.

Developing high performance computing programs requires, in addition to the normal sequential software development requirements, knowledge of the high performance computing architecture details. For vector and parallel program development, this means data or control parallelism, partitioning and mapping, and implementation details need to be considered. Other factors such as communication, data layout, process synchronization and thread control may also play a role. It is almost a cliché to describe vector and parallel programming as difficult, time consuming and error-prone. Exacerbating the situation is the existence of many legacy codes together with those written by programmers with widely varying backgrounds and experience.

Understanding these kinds of programs helps to: identify code fragments that could be optimized; understand the overall algorithms or techniques used; and, enhance, maintain and integrate existing code into new operational environments. However, program comprehension of vectorized or parallelized code is more difficult than than of sequential programs due to the varied architecture, granularity issues, and partitioning and mapping issues.

Visualization of concepts inherent in the program code is proposed as a new mechanism to facilitate program comprehension of high performance computing codes. 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. In this research, relations that express the semantic connections between two individual program statements are established. More abstract relations connecting lower-level relations are developed in a hierarchy whereby the semantic information is propagated in more abstract notions upwards through the hierarchy. Concepts are then defined by the high level abstractions in this relational hierarchy, that is, concepts represent abstractions of code fragments.

The focus of this paper is to apply these ideas about concept visualization to a parallel computing program. A case study approach is adopted. A simple MPI parallel program is analyzed. Conceptual information about the program is identified and extracted. Several visualizations of the extracted information are presented and discussed.

This paper is organized as follows. Proposed concept vi-

sualizations suitable for high performance computing codes are described in further detail in Section 2. A case study is introduced in Section 3. Conclusions are given in Section 4.

2 Concept Visualization

Concept visualization appears to have several broad definitions in the literature. In some cases, the term is used as a noun and refers to the visualization model used or the data model that is to be visualized. In other cases, it refers to visualizations aimed at facilitating the viewer's understanding about the visualized data. In fact, this is the purpose of visualization itself. There is some research that relates to the visualization of concepts itself. One interesting application of concept visualization is that in [8] where a "... viewer may follow the [concept] map which branches off to various nodal points. Each of these nodal points represents a single theory." An application of superquadrics and conceptual spaces for robotics is discussed in [2]; an application for intrusion detection in computer security is presented in [1]. The use of superquadrics also enables visualizations of these conceptual domains.

This research is also strongly related to Program Visualization, a sub-field of Information Visualization. Program Visualization seeks to present visual information about the program, often, concentrating on run-time events. Pretty-printing code is a trivial example of an old type of textual code visualization. Now-a-days, graphical representations are most often used. Most of the work that combines concept visualizations and program visualizations appear in the context of computer science curriculum and student learning. Here, the educational goal is to have the students learn about the algorithms and data structure changes that occur during a program's execution.

Visualization is an important aspect of high performance computing. Many computational science applications make use of visualization to provide graphical displays associated with numerical simulation or experimental analysis. Visualization has also been suggested for use in petascale computing together with the exploration of large, tera- and petabyte datasets [5]. Visualization has also been used for task allocation, communication, data dependency analysis and debugging. For the most part, these visualizations facilitate identification of specific events or behavior associated with run-time analysis of programs, often, for purposes of fine-tuning the performance.

Concept visualization for high performance computing programs specific to facilitate the reader's understanding of the concepts inherent in the code has not been considered before. The program concepts that are of interest here and which are applicable to high performance computing include: the identification of the purpose(s) of the code and sub-purposes of code fragments; identification of computational and communication components; and identification of vectorized or parallelized code fragments, or potential code fragments.

The premise of this research is that high performance computing program comprehension requires the reader to form concepts about the purpose and intended actions of programs. However, since present day mechanisms do not facilitate this process, an approach to do so is developed in this section. Traditionally, the reader considers input from the textual presentation of the code as well as from the 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 poor or incorrect, the volume of available information requires time to assimilate, the code itself may be hard to read, and there is added complexity in the implementation due to the high performance computing requirements.

The 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 following set of proposed heuristic rules are established to guide the relational hierarchy organization.

1. Data dependency of write-write, read-write, write-read or read-read are always represented as Level 2 relations.

2. Repetition always forms Level 2 relations between the

construct header/footer and each and every statement in its body.

3. Conditional selection always forms Level 2 relations between the construct header and each and every statement in its body.

4. A function/procedure/object/task header both participates in dependency analysis as well as forming Level 2 relations with each and every statement in the function.

5. A function/procedure/object/task call always is a relation with one child as the Level 1 call, and the other child as the concept relation pertaining to the function code.

The Advanced Relation Model for Program Visualization (ARM 4 PV) [3] provides the framework for the proposed concept visualization technique herein. The technique is two phased. In the first phase, program code is analyzed so as to determine the relation hierarchy. 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 [6], and the Program-Scientific Visualization (PSV) model [9]. The CCV and PSV models are sub-models in the ARM 4 PV.

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, $\overline{(S_i,0),(S_i,1)}$. 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: $\overline{(x_1,i)(x_2,l)}$, $\overline{(x_2,l)(x_3,j)}$ 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. Line structured visualizations are selected in this paper.

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 [9]. 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

$$R_1^1 = (S_1, d_1, (e_{1,1}, e_{1,2}, e_{1,3}))$$
$$R_2^1 = (S_2, d_2, (e_{2,1}, e_{2,2}, e_{2,3}))$$

then

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

where $v = \left(\frac{(e_{1,1}+e_{2,1})}{2}, \frac{(e_{1,2}+e_{2,2})}{2}, \frac{(e_{1,3}+e_{2,3})}{2}\right)$ and 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.

3 Case Study

The study focuses on the introductory textbook example of the MPI parallel program to calculate the value of π using numerical integration [7]. The program source code with statements prefixed by line numbers appears in Figure 1. A sample of the relation database used in this visualization appears in Figure 2. The last line, $R_1^5 = (R_1^4, R_{47}^1, \text{Communicate aggregate area})$ specifies that the highest level relation of Level 5 in this example has the semantics 'Communicate aggregate area' (the semantics in the figure are hand-crafted since the current prototype does not provide for a semantic engine to abstract new semantics for higher level relations).

The relation hierarchy for this example is given in Figure 3. For this case study, and due to the fact that only five levels of abstraction are defined, Level 3 relations may be considered as low-level concepts. The first one, R_1^3 describes the low-level concept of communicating interactive user input (see Figure 1), i.e., a `printf` statement followed by a `scanf` statement combined with the broadcast in Statement 35. This concept means that the user input is communicated to all the processors. This is the meaning of R_1^3. In terms of the attribute vector, the second level relation is updated as: $R_{77}^2 = (R_{31}^1, R_{32}^1, \text{"interactive user input"}, (0,0,1))$ which indicates that the abstraction is 100% input/output. R_1^3's associated vector is $(0, \frac{1}{2}, \frac{1}{2})$

```
 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 = 3.14159265358979323846264643;
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,
                           MPI_COMM_WORLD);
48
49              if (rank== 0) {
50                  printf("pi is approximately %.16f, Error is %.16f\n",
51                         pi, fabs(pi - PI25DT));
52                  endwtime = MPI_Wtime();
53                  printf("wall clock time = %f\n",
54                         endwtime-startwtime);
55              }
56          }
57      }
58      MPI_Finalize();
59      return 0;
60  }
```

Figure 1. The cpi.c MPI example as the program to visualization: this calculates pi by numerical integration in parallel.

which indicates that this abstraction equally supports communication and input/output. Some of the concept visualizations described subsequently visualize these relations.

Figures 4, 5 and 6 show line structured CCV conceptual visualizations of the program in Figure 1 as represented through the relation hierarchy. The relation $R_1^2 = (R_7^1, R_9^1$,Function f) is the small hat-shaped region on the left-hand side of the figures. 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 4 shows a non-glyph highlighted image whereas Figures 5 and 6 show glyph highlighted images. Figure 5 is a zoomed-in image where the glyphs highlight the concepts that are of interest. The glyphs in Figure 6 highlight the relations that participated in the formation of the concept relations (note, the glyph coloring may not show all of the relations that it participates in). (The colors used in these figures highlight the different concepts; gray-shaded reproductions also provide these highlights.)

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 that makeup the main function. The `while`

$R_7^1 = (S_7$, Function f header)
$R_9^1 = (S_9$, Compute)
$R_{30}^1 = (S_{30}$, Conditional)
$R_{31}^1 = (S_{31}$, Output)
$R_{32}^1 = (S_{32}$, Input)
$R_{33}^1 = (S_{33}$, Access timer)
$R_{35}^1 = (S_{35}$, Broadcast)
$R_{41}^1 = (S_{41}$, Repetition)
$R_{42}^1 = (S_{42}$, Compute)
$R_{43}^1 = (S_{43}$, Compute)
$R_{45}^1 = (S_{45}$, Compute)
$R_{47}^1 = (S_{47}$, Reduce)
$R_{49}^1 = (S_{49}$, Conditional)
$R_{52}^1 = (S_{52}$, Access timer)
$R_{53}^1 = (S_{53}$, Output)
$R_1^2 = (R_7^1, R_9^1$, Function f)
$R_{57}^2 = (R_{30}^1, R_{32}^1$, Conditional input)
$R_{72}^2 = (R_{41}^1, R_{42}^1$, Compute multiple x values)
$R_{73}^2 = (R_{41}^1, R_{43}^1$, Aggregate function values)
$R_{76}^2 = (R_{49}^1, R_{53}^1$, Master printing)
$R_{77}^2 = (R_{31}^1, R_{32}^1$, User input)
$R_{79}^2 = (R_{33}^1, R_{52}^1$, Obtain timings)
$R_1^3 = (R_{35}^1, R_{77}^2$, Comm user input)
$R_2^3 = (R_1^2, R_{73}^2$, Repeated function f call)
$R_3^3 = (R_{57}^2, R_{77}^2$, Master user input)
$R_4^3 = (R_{79}^2, R_{76}^2$, Master printing timings)
$R_5^3 = (R_{72}^2, R_{73}^2$, Aggregate function values for multiple x inputs)
$R_1^4 = (R_3^3, R_{45}^1$, Compute aggregate area)
$R_1^5 = (R_1^4, R_{47}^1$, Communicate aggregate area)

Figure 2. Excerpt from the relational hierarchy.

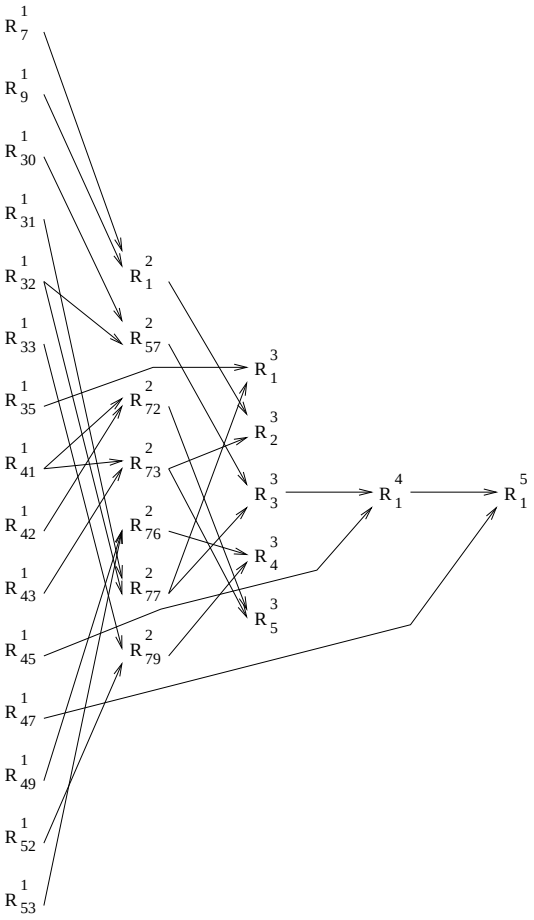

Figure 3. Hierarchal organization of higher-level relations in the relation database

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 5.

- 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 lexico-

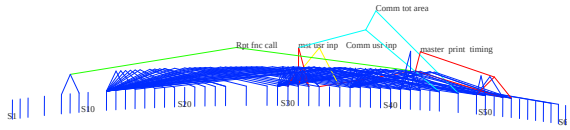

Figure 4. Concept visualization: non-glyph highlighted with text notations.

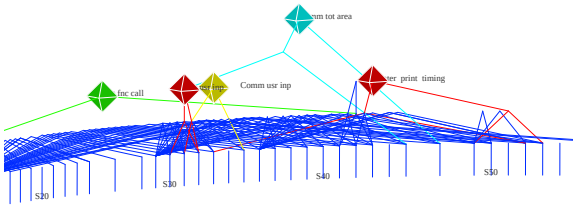

Figure 5. Concept visualization: zoomed, glyph highlighted with text notations.

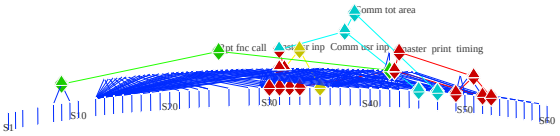

Figure 6. Concept visualization: fully glyph highlighted with text notations.

graphic 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 1 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.

Figures 7, 8 and 9 show the application of the PSV model as 3-D plot of the attributes in the relation hierarchy pertaining to the program in Figure 1. Recall that the attributes of interest in this study are computations, communication

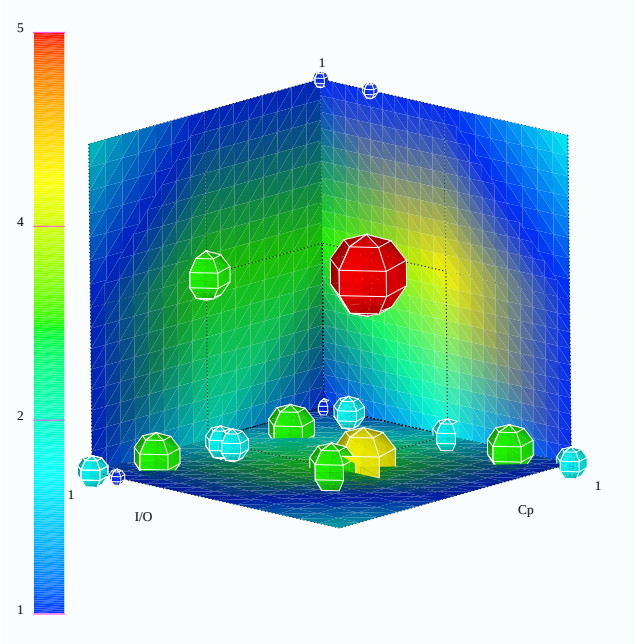

Figure 7. Concept visualization: 3D view.

and input/output, specifically, the extent to which these attributes apply to the relations and in particular, the concepts. The axes in these figures are labeled "Cp", "Cm" and "I/O", and refer to computation, communication and input/output, respectively. 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 7 is displayed with a slight forward tilt, the "Cm" axis is vertical in this figure. Figure 8 is an orthogonal view with depth cues showing input/output versus computations, while, Figure 9 is an orthogonal view with depth cues showing 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 7, suggests a communication of an input/output value.

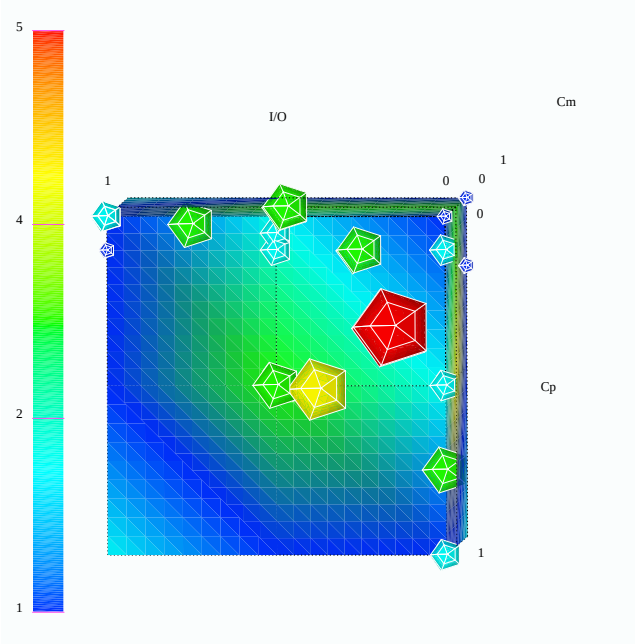

Figure 8. Concept visualization: 2D orthogonal view with depth cues.

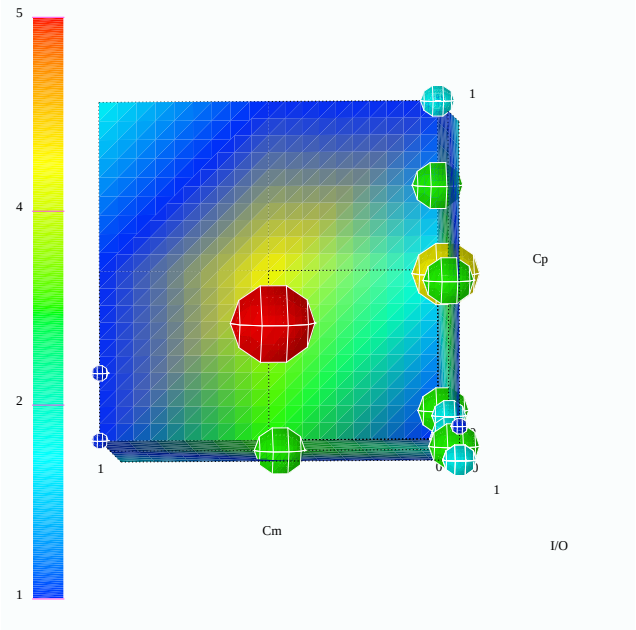

Figure 9. Concept visualization: 2D orthogonal view with depth cues.

- 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.

These observations from the visualization show that the concepts of parallel programming in terms of its expected computational, communication and input/output behavior can be extracted from the visualization. In this regard, the PSV model visualization appears successful in the *quantification* of the parallel program.

4 Conclusions

There is a need to better understand high performance program codes, especially, in an environment where the codes reflect the diversity in architecture as well as in programming and language models. Legacy codes and those written by individuals with varied backgrounds further diversify the code base. The interests in porting high performance code in new implementation environments also reflect on the need to understand high performance codes. Parallel programs represent a large body of such high performance computing programs.

Conceptual visualization is proposed in this paper to facilitate reading and understanding parallel programs. A relational model consisting of a hierarchy of binary relations between program statements together with higher-order binary relations between lower level relations provides for conceptual abstraction representation about code fragments. The relation hierarchy is organized such that there is an up-wards propagation of abstractions, thereby, leading to highly abstract descriptions about the program. Such abstract descriptions are termed concepts in this work. A visualization technique based on the relation hierarchy and composed of two distinct visualization models is further proposed in this paper.

A study of a simple MPI parallel program is made. Various visualizations from both of the proposed visualization models are included and discussed. Combined, the visualizations provide for qualitative and quantitative aspects relating to the concepts in the given program. In particular, this study addresses two of the three stated interests of Section 2, namely, that the purpose(s) as well as the computational and communication aspects of code fragments can be visualized by the proposed techniques. This study suggests that the proposed visualizations do facilitate the understanding of parallel programs.

Other applications of the proposed models together with an overview of the MPI study of this paper are discussed in [4]. The additional applications include the visualization of a simple sequential program. In addition, brief comments regarding the visualization work-in-progress of a FORTRAN code are made. The FORTRAN program consists of recurrence equations to calculate the mean and variance of the gain in Avalanche Photo Diodes with the inclusion of the dead-space effect. This work is aimed at the parallelization of the code.

There are several significant limitations of the current visualization system and models. First, the success of the visualization system is dependent on the accuracy and completeness of the relation hierarchy. The current prototype system, and which this study is based on, does not provide for the semantic propagation in the hierarchy. Thus, this was hand-crafted for this study, and, although representative of the program related information, lacks completeness. An automated semantic engine together with relation accuracy verification is required in the future. The visualization models themselves are well developed in prior work. However, assessment of the visualizations has not yet been completed. Lastly, the prototype needs to be extended and further developed.

References

[1] 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.

[2] A. Chella, S. Gaglio, and R. Pirrone. Conceptual representations of actions for autonomous robots. *Robotics and Autonomous Systems*, 34:251–263, 2001.

[3] 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).

[4] B. J. d'Auriol. Concept visualizations of computer programs. In *Proc. of The Conference on Advances in Internet Technologies and Applications (CAITA 2004)*, Purdue University, West Lafayette, Indiana, USA, July 2004. (to appear).

[5] J. Dongarra and D. Walker. The quest for petascale computing. *Computing in Science & Engineering*, 3(3):32–39, May-June 2001.

[6] 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.

[7] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI, Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, 1999.

[8] 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*, pages 233–240, London, UK, July 2001.

[9] R. Policherla. Scientific visualization techniques for program visualization. Master's thesis, Department of Computer Science, The University of Texas at El Paso, May 2004.