STView: An Eclipse Plug-in Tool for Visualizing Program Structures in Fortran Source Codes

Tomomi Ohichi*, Masaaki Terai†, Mitsuo Yokokawa*† and Kazuo Minami†
*Kobe University
†RIKEN Advanced Institute for Computational Science (AICS)

I. ABSTRACT

We developed an Eclipse plug-in tool named STView for visualizing the program structures of Fortran source codes to help improve the performance of the program on a supercomputer. To create a tree that represents program structures, STView uses an abstract syntax tree (AST) generated by Photran and filters the tree because the AST has many nontrivial nodes for tokens. While ordinary visualization tools such as profiler and refactoring tools represent a call tree that only includes the relationship among procedures, STView can visualize loops and branches in addition to a call tree. Moreover, STView can show the time-consuming parts or hotspots of a program by profiling data. We evaluated the capability of STView using 13 scientific applications collected from websites and confirmed that it robustly visualizes a tree for all applications.

II. INTRODUCTION

In the past decade, various applications used on petascale supercomputers were developed to elucidate scientific and engineering issues using computer simulations. Many kinds of experts as well as code developers work together to develop simulation codes and efficiently perform the simulations on modern high-performance computing systems. To improve application performance, they need to understand the program structure before they modify the source codes. However, to achieve more realistic simulations, modern applications have become more complicated, have more source modules than those previously used, and usually provide insufficient documentation. For an engineer who is not a code developer, the task of reading codes to understand program structures requires much effort and time. In this situation, the development of an analysis tool for visualizing program structures in the form of a tree is important.

There are a few analysis tools that visualize program structures of Fortran source codes. However, a non-proprietary tool usually does not support Fortran 90 or higher source codes and only provides a call tree that includes the relationship among procedures, even though we require program structures including loops and branches, which are the time-consuming parts or hotspots of code. In other words, most conventional tools are not constructed for the purpose of improving application performance.

To create a tree that represents program structures, a source code analysis tool usually uses an AST generated by a parser. However, the Fortran programming language imposes complicated lexical and syntax rules and allows vendors to employ their own specifications or dialects. To address these difficulties, we employed Photran [1], an Eclipse plug-in, to develop and refactor Fortran 77–2008 source codes. Although Photran is not an analysis tool that improves application performance as much as other tools, it satisfied our objectives because it has the capability to visualize program structures and provides the fundamental features to perform source code analysis. Therefore, we developed a tool named STView for visualizing the program structures of Fortran source codes using Photran's parser. Moreover, we implemented various features such as tree filtering, tree printing, and importing profiling data for Fujitsu FX10 in order to improve the usability of STView. Finally, we evaluated STView's ability to visualize program structures with various applications.

III. RELATED WORK

A few non-proprietary analysis tools for Fortran code are available [2], [3]. However, since they are usually designed to detect syntax or semantic errors for refactoring, most of them provide a call tree without precise program structures that include loops and branches. Moreoever, they only support FORTRAN 77 or limited features of Fortran 90 specifications.

K-scope [4] is a Fortran 77/90 source code analysis tool developed by Terai et al. To simplify the K-scope design, it uses an external parser called the Omni XMP compiler to create intermediate codes. However, its parser employs a one-pass operation, whereas other commercially available parser usually performs a multi-pass operation to resolve complicated codes written in the Fortran programming language. Because the Omni XMP compiler is currently under development, the parser often fails to translate source codes to intermediate codes in scientific applications. Extending practical use of K-scope appears to be difficult owing to the above situation.

IV. FEATURES AND IMPLEMENTATION

A. Features

STView comprises two views: a tree view and a profiler view in addition to conventional views such as the editor and the project explorer provided by Eclipse and Photran.

The tree view shows a program structure in the form of a tree, each node of which stands for a loop, branch, and procedure call in the program; therefore, users can readily grasp a view of the program flow. Moreover, STView searches

the nodes of the tree by regular expressions (e.g., call mpi_[a-zA-Z]+) and shows the results highlighted in the tree view. Each node in the tree is linked to a corresponding part of the source code so that users can immediately jump to that part on the editor. Furthermore, the generated tree can be printed and exported as a file in the PDF format.

The profiler view shows the time-consuming parts or hotspots using profiling data measured in the sample-based profiler for Fujitsu FX10.

B. Implementation

STView is implemented using Photran ver. 9.0.1, Eclipse ver. 4.4.1, and Java Runtime Environment ver. 1.8, and performs the following three steps to visualize a tree:

- creates a list of Fortran filenames,
- creates an original AST from Photran's AST for each program unit, and
- visualizes of a tree of program structures represented by the original AST.

First, STView obtains the opened files in the editor of Eclipse, collects filenames of Fortran source codes in folders including the opened files, and creates a list of filenames.

After creating the list, Photran's parser generates ASTs for each program unit based on the filename list. Because Photran's AST includes many non-trivial tree nodes, STView creates a new STView-AST that comprises only loops, branches, and procedure calls to reduce the number of tree nodes. During the translation process from source codes to STView-ASTs, STView ignores dependencies among program units (e.g., program, subroutine, and function) and does not use a building script (e.g., Makefile). After all the STView-ASTs are set up, STView creates a single tree of the entire program based on node identifiers as shown in Fig.1. The process is very simple and the built tree satisfied our objectives to understand a brief sketch of the code.

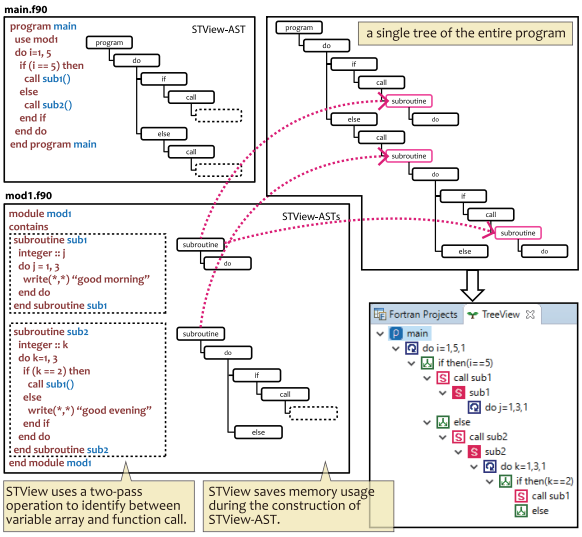

Fig. 1. Construction of a tree from STView-ASTs.

TABLE I
APPLICATIONS THAT ARE USED FOR THE EVALUATION.

Name	Description	File ext.	#files	SLOC
AxiSEM-1.1	Spectral element method	F90+f90	80	52690
calypso	Magnetohydrodynamics	F90+f90	694	174298
cftb	Crystal field tight binding	f+f90	189	55244
evora	Cosmological evolution simul.	f90	23	5504
FDS-6.5.1	fire-driven CFD (LES)	f90	35	144268
gtc	Gyrokinetic toroidal simul.	F90	32	12809
MODYLAS-1.0.2	Molecular dynamics	f	41	40399
MOPAC-7.1	Quantum chem. (MO)	F90+f90	590	64132
NICAM-DC-1.1	Global climate simul.	f90	130	55300
RSDFT-1.2.2	Real-space DFT	f+f90	212	62656
SHF_OMP	Pseudospectral method	f90	30	37491
SMASH-1.1	Quantum chem.	F90	33	75934
vef	Radiative tranfer eqn. solver	f90	34	1585

Because Fortran uses the same token for both array variables and function calls, the difference between them cannot be recognized by lexical analysis in a single line of code. STView employs a two-pass operation to address this problem. In the first-pass operation, STView collects function names defined by a user and creates a list. Moreover, to prevent excessive memory consumption, after each search of a program unit, STView deallocates Photran's objects that can be used for AST. Using this deallocation method, tree construction is faster than that using the other method which does not deallocte the objects. In the second-pass operation, STView identifies a function call based on both the created list and intrinsic procedure table provided by Photran.

Finally, STView visualizes a tree based on the STView-AST.

V. EVALUATION AND DISCUSSION

To evaluate how STView visualizes program structures, we considered 13 scientific applications from websites as shown in Table I. Moreover, prior to this evaluation, we preprocessed the source codes with the capitalized file extensions (e.g., *.F, *.F90) because the current version of STView cannot handle directives (e.g., #if, #ifdef).

STView created visual program structures for all applications without any critical errors. Because the current version of STView did not support the interface statement in Fortran. STView ignored part of the source code and approximately visualized program structures in the following applications: "AxiSEM", "cftb", "FDS", "gtc", "MODYLAS", and "MOPAC." However, we found that STView could visualize program structures for the applications even though they had the interface statement in their source code. We confirmed that STView is useful to understand program structures and shows much promise for practical use; however, to improve its usefulness, we need to resolve this interface-statement problem.

REFERENCES

[1] Photran. http://www.eclipse.org/photran/.
[2] FTNCHECK. http://www.dsm.fordham.edu/ftnchek/.
[3] Source Navigator. http://sourcenav.sourceforge.net/.
[4] M. Terai at el. Extending K-scope fortran source code analyzer with visualization of performance profiling data and remote parsing of source code. In *2014 IEEE Inter. Conf. on High Perf. Comp. and Comm.*, pages 866–873, 2014.