CCA/EBT: Code Comprehension Assistance Tool for Evidence-Based Performance Tuning

M. Hashimoto[†*], M. Terai[*], T. Maeda[†], and K. Minami[*]

[†]Software Technology and AI Research Lab, Chiba Institute of Technology
[*]RIKEN Advanced Institute for Computational Science

INTRODUCTION

Application performance tuning is still quite an art, despite advances in auto-tuning systems [1][9].

EBT (evidence-based performance tuning) [5] aims at helping performance engineers gain and share evidence of performance improvement to make better decisions.

Long-term goal is to construct a database of facts, or ***factbase***, extracted from performance tuning histories of computational kernels such that we can search the database for promising optimization patterns that fit a given computational kernel.

OBJECTIVES

- Locating computational kernels
  - Predicting location of computational kernels
  - Assisting in the manual inspection of source code
- Identifying optimization patterns applied to computational kernels
- Constructing database of positive/negative examples of optimization patterns

TECHNICAL HIGHLIGHTS

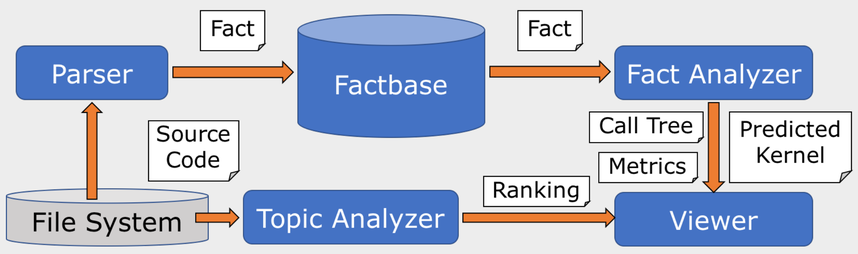

Fig. 1: Overview of CCA/EBT

Loop Kernel Prediction based on Machine Learning

1) Features were extracted from 175,963 loops from 1000 computation-intensive applications hosted on GitHub [6].
2) 100 were randomly sampled and then manually classified by experienced performance engineers.
3) By using the classification results as training data and C-SVC in LIB-SVM [3] from scikit-learn, we constructed a predictive model.
4) The model achieved 20-fold cross-validated classification accuracy of 81% [6].

Tab. 1: Syntactic features of a loop

Abbrv.	Feature
FOp	# floating-point operations
St	# statements
Br	# branches
AR	# array references
DAR	# direct array references
IAR	# indirect array references
B/F	Bytes per flop
MLL	Maximum loop nest level

Dedicated Fortran Parser

Specifications: FORTRAN77, Fortran90, Fortran95, Fortran2003, Fortran2008
Dialects: IBM, PGI, Intel
Directives: Cpp, OpenMP, OpenACC, OCL(Fujitsu), XLF(IBM), DIR/DEC(Intel)

Schemes for Statically Estimating Volume of Memory Traffic

ES0 Data is shared in cache only among syntactically identical array references.
ES1 The data referenced by the array references that differ only by the first dimension are located in the same cache block. (ex. `a(i, n)` and `a(j, n)`)
ES2 The data referenced by the array references that differ only by the first dimension and by additions/subtractions of constants at the second dimension are located in the same cache block. (ex. `a(i, n)` and `a(j, n + 1)`)

Topic Analysis for Source Code

- Helping performance engineers understand an application
  - Analyzing comments and variable names occurring in the source code
    - Examining the topic or research field of the application.
- Constructing a topic model with *latent semantic indexing* (LSI) [4]
  - Based on 168 papers of scientific applications from several research fields
    - Quantum chemistry, astrophysics, climate science, ...

RELATED WORK

Commercial and open-source Fortran analysis tools include the following:

- FORCHECK [2] --- A Fortran source code analyzer and programming aid,
- Photran [7] --- An IDE and refactoring tool for Fortran, and
- CamFort [8] --- Light-weight verification and transformation tools for Fortran.

CCA/EBT is capable of predicting loop kernels and of parsing 1000 applications in a fully automated way.

VIEWERS

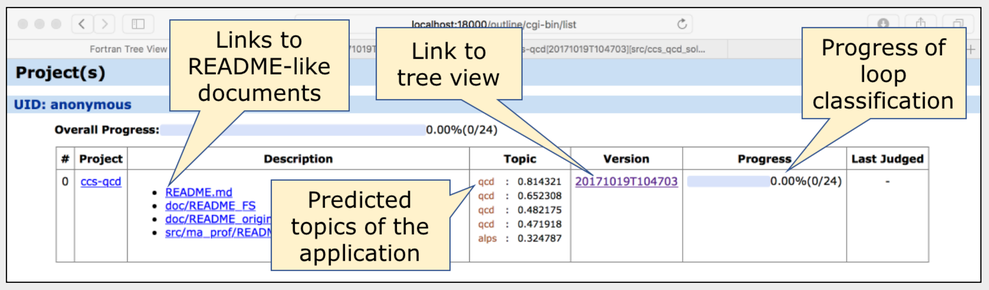

Fig. 2: Project view

A project summary view provides the following:
- Automatically generated links to the documents whose names contain “README”,
- The result of topic analysis as a ranking of candidate applications (qcd, alps, ...),
- A link to the tree view, and
- Progress of a user’s loop classification performed in the tree view.

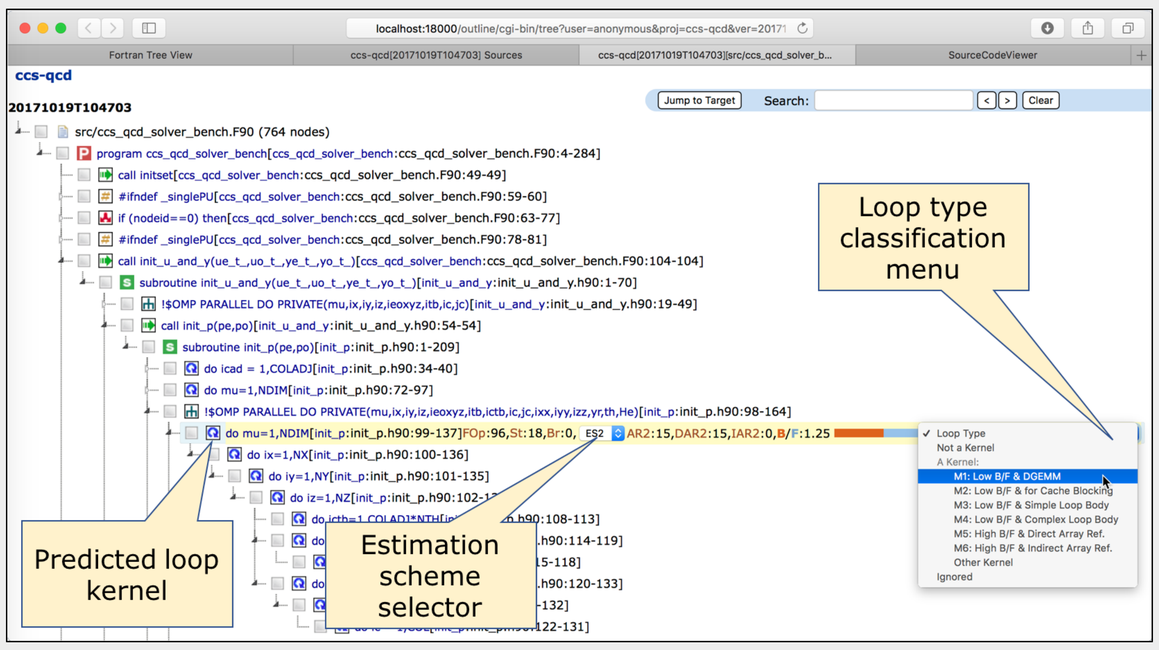

Fig. 3: Tree view

A tree view shows the following:
- Outline of the AST and the call tree of an entire application,
- Predicted loop kernels decorated with extracted static features (estimated B/F, ...), and
- Estimation scheme selector for features AR, DAR, IAR, and B/F
  (suffixes 0, 1, and 2 of them indicate the estimation schemes ES0, ES1, and ES2, respectively).

⚠ The size of a call tree may be infinite when it contains recursive calls.
A procedure or a function appear only at the deepest level of non-recursive calls.

A source code view (appears by double-clicking a tree node) has the following:
- Highlighted array references,
- Quick look of the definitions (appear on mouse-over),
- The definition of an array reference (by double-clicking).

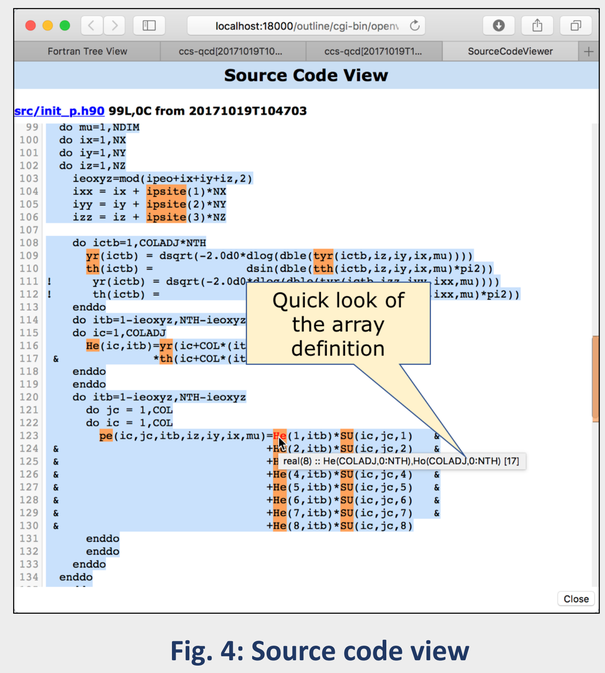

Fig. 4: Source code view

The tool is available at
https://github.com/ebt-hpc/cca

REFERENCES

1. Basu et al. *J. High Perform. Comput. Appl.* **2013**, 27, 4.
2. Forcheck b.v. http://www.forcheck.nl/.
3. Chang & Lin. *Trans. Intell. Syst. Technol.* **2011**, 2, 3.
4. Deerwester et al. *J. American Society for Info. Sci.* **1990**, 41, 6.
5. Hashimoto et al. *Proc. Conf. on Mining Software Repositories*. **2015**.
6. Hashimoto et al. *Proc. Conf. Performance Engineering*. **2017**.
7. Johnson et al. http://www.eclipse.org/photran/.
8. Orchard & Rice. *Proc. Workshop on Refactoring Tools*. **2013**.
9. Tiwari et al. *J. High Perform. Comput. Appl.* **2011**, 25, 3.
10. Williams et al. *Commun. ACM*, **2009**, 52, 4.

ACKNOWLEDGMENTS

This work was supported in part by JSPS KAKENHI Grant Number JP26540031.