Automated analysis of algorithms implemented on top of TBD

Walter Tichy, Umut Acar, Emanuel Jöbstl
Karlsruhe Institute of Technology, Faculty of Computer Science
Carnegie Mellon University, Computer Science Department

***Abstract*—The principle of incremental programming has been well known for many years. While the autmation transformation of imperative programs into fast incremental programs is not solved in general, a number of platforms for incremental computation have been created. In this document, we inspect the incremental computation platform TBD. First, we summarize known principles like directed depenedency graphs, execution traces, intrinsic trace distance and trace stability, then we create an program model which enables us to apply those principles to TBD programs. We also present an algorithm to calculate the intrinsic trace distance in practice and how we can make automatic optimizations to prorgrams by analyzing the dependency graph.**

Keywords—Incremental computing, tbd, scala

I. Introduction

This section should describe the purpose of incremental computation and also explain for which purpose the concept of incremental computation is useful. Also, used and referenced terminology should be outlined.

A. *Approaches to incremental computation*

This subsection should briefly describe various approaches to incremental computation, and outline their strengths and weaknesses. Also, some newer implementations of incremental computation platforms should be outlined. [1] [2] [3] [4] [5] [6] [7] [8] [6] [9] [10] [11] [**?**]

II. Theoretic funcdamentals

This section should give a brief summarization of the principles described in the work of U. Acar et al. This section is important, because we later have to adjust definitions found here to fit the program model of TBD. [12]

A. *Theoretical program model*

This subsection should describe the underlying theoretical program model.

B. *Approach for incremental computation*

This subsection should describe the approaches of using Traces, DDGs (Directed Dependency Graphs) and memoization together to accomplish incremental computing.

C. *Stable algorithms*

This subsection should describe the concepts of stable algorithms, intrinsic trace distance and their relationship.

It should especially be made clear, that the intrinsic trace distance forms a lower bound for the time needed by change propagation during an update.

III. TBD

This section should outline the TBD framework, a framework for incremental computation currently being developed at CMU. The framework is being developed in the Scala language. This enables us to exploit the relection capabilties of Scala for analysis [**?**].

A. *Programming interface*

This subsection should describe the TBD core API. Since TBD relies on the use of patterns known from functional languages, these patterns should be described. Also, the constraints for and the responsibilites of the developer have to be specified. The section should be concluded with an example.

B. *Program model*

This subsection should describe a program model for TBD, which can be used for theoretical conclusions in the following sections. The differences and similarities with the programming model in section II should be highlighted.

IV. An Intrinsic trace distance algorithm for TBD

This section should describe an algorithm for calculating the minimal trace distance of traces produced by TBD. Also, implementation details and challenges during the implementation (like comparing functions for equality) should be discussed. The definition of intrinsic trace distance can be found in [12], chapter 7.

A. *Proof of correctness*

We shall also proof the correctness of our algorithm. It is of importance that we take the changes in the program model into account.

V. Automatic optimization of algorithms

This section should describe how analyzing the DDG can be used for automatic optimization, or at least to give hints for possible optimization to the developer.

The exact contents of this chapter are still to be determined, based on our findings. Possible approaches include, but may not be limited to:

- Operation reordering.
- Insertion of memorization commands.
- Detection of cascading updates, which could be omitted.

VI. Evaluation

This section should demonstrate the usefullness of the described techniques using real-world algorithms, like map, reduce and quicksort.

Basically, it should be demonstrated how it is possible to optimize a classic implementation (without memoization) of each algorithm, so that change propagation time lies within the same complexity class as the theoretical lower bound for updates for this algorithm.

VII. Conclusion

This section should conclude and summarize with the findings of this work.

A. Future work

The final section should shortly outline problems encountered but not solved during the writing of this thesis, as well as encourage future research on interesting issues of incremental computation.

References

[1] A. Heydon, R. Levin, and Y. Yu, "Caching function calls using precise dependencies," *ACM SIGPLAN Notices*, vol. 35, no. 5, pp. 311–320, 2000.

[2] U. A. Acar, G. E. Blelloch, R. Harper, J. L. Vittes, and S. L. M. Woo, "Dynamizing static algorithms, with applications to dynamic trees and history independence," in *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2004, pp. 531–540.

[3] U. A. Acar, G. E. Blelloch, and R. Harper, "Adaptive functional programming," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 28, no. 6, pp. 990–1034, 2006.

[4] R. F. Cohen and R. Tamassia, "Dynamic expression trees and their applications," in *SODA*, 1991, pp. 52–61.

[5] Y. A. Liu and T. Teitelbaum, "Systematic derivation of incremental programs," *Science of Computer Programming*, vol. 24, no. 1, pp. 1–39, 1995.

[6] F. McSherry, R. Isaacs, M. Isard, and D. G. Murray, "Composable incremental and iterative data-parallel computation with naiad," Tech. Rep. MSR-TR-2012-105, October 2012. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=174076

[7] Y. Chen, U. A. Acar, and K. Tangwongsan, "Functional Programming for Dynamic and Large Data with Self-Adjusting Computation," 2014. [Online]. Available: http://www.mpi-sws.org/~chenyan/papers/icfp14.pdf

[8] W. Pugh and T. Teitelbaum, "Incremental computation via function caching," in *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1989, p. 315328. [Online]. Available: http://dl.acm.org/citation.cfm?id=75305

[9] M. A. Hammer, U. A. Acar, and Y. Chen, "CEAL: a C-based language for self-adjusting computation," in *ACM Sigplan Notices*, vol. 44. ACM, 2009, p. 2537. [Online]. Available: http://dl.acm.org/citation.cfm?id=1542480

[10] R. Ley-Wild, M. Fluet, and U. A. Acar, "Compiling self-adjusting programs with continuations," in *ACM Sigplan Notices*, vol. 43, no. 9. ACM, 2008, pp. 321–334.

[11] D. Peng and F. Dabek, "Large-scale Incremental Processing Using Distributed Transactions and Notifications." in *OSDI*, vol. 10, 2010, p. 115. [Online]. Available: https://www.usenix.org/legacy/events/osdi10/tech/full_papers/Peng.pdf?origin=publication_detail

[12] U. A. Acar, "Self-adjusting computation," 2005.