Refactoring for parallelization

Tamás Kozsik

Department of Programming Languages and Compilers
Eötvös Loránd University (ELTE), Budapest, Hungary
Tamas.Kozsik@elte.hu

The development of efficient parallel programs requires significant effort from the programmer's side. On the one hand, due to the increased complexity, it is much harder to guarantee the correctness of the parallel code than that of the sequential one. On the other hand, it is not straightforward to find out which is the best way to parallelize a given program with a given input on a given hardware architecture. Often there are many possibilities, and the software developer needs to experiment with the code, and to apply profiling techniques, in order to make a good parallelization decision. It may even turn out that there is no effective parallelization of the given code under the circumstances, because the runtime overhead of any parallelization cancels out all performance gains. In such cases the programmer's efforts were just wasted.

In order to minimize the effort put into parallelization and experimentation, software developers may be willing to use a refactoring tool to easily, quickly, and – most importantly – safely apply the source code transformations required by the insertion and removal of parallel programming constructs. Refactoring is the process of changing the code without changing its observable behaviour. The goal of refactoring is to modify some non-functional properties of the code, e.g. to improve its quality. A refactoring tool offers semantics preserving code transformations, and can help avoid making errors during the process of refactoring. Since parallelization can be regarded as refactoring, some refactoring tools (e.g. [4]) have already started to offer parallelization transformations.

Software development tools can not only automate code transformations, but they can also facilitate the identification of parallelization opportunities. Pattern candidate discovery [3] is a static source code analysis technique, which can find source code fragments amenable to refactoring into instances of common parallel patterns. Parallel patterns [2] describe the sequential/parallel structure of computations at a high level of abstraction. They can be implemented using algorithmic skeletons [1], which are reusable, composable and configurable structures for parallel computations.

This talk gives an overview of the refactoring techniques to support pattern-based parallelization, and describes recent achievements of code *paraphrasing* [2] in various functional programming languages. The talk has been supported by the European Union, and co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013).

References

[1] Cole, M., *Algorithmic Skeletons: Structured Management of Parallel Computation*. MIT Press, 1991. ISBN 0-262-53086-4

[2] Hammond, K., Aldinucci, M., Brown, C., Cesarini, F., Danelutto, M., González-Vélez, H., Kilpatrick, P., Keller, R., Rossbory, M., Shainer, G., The ParaPhrase Project: Parallel Patterns for Adaptive Heterogeneous Multicore Systems. *In: Formal Methods for Components and Objects*, LNCS 7542, 2013. pp. 218–236.

[3] Kozsik, T., Tóth, M., Bozó, I., Horváth, Z., Static analysis for divide-and-conquer pattern discovery. *Computing and Informatics* 35(4):764–791, 2017.

[4] Kozsik, T., Tóth, M., Bozó, I., Free the Conqueror! Refactoring divide-and-conquer functions. *Future Generation Computer Systems* 79(2):687–699, 2018.