Divide a Divide-and-Conquer into a Divide and a Conquer

Tamás Kozsik, Melinda Tóth, István Bozó

Dept. Programming Languages and Compilers, Eötvös Loránd University, Hungary
{tamas.kozsik,tothmelinda,bozoistvan}@elte.hu

Abstract

Divide-and-conquer algorithms appear in the solution of many computationally intensive problems, and are good candidates for parallelization. Their identifying property is that such an algorithm divides its input into "smaller" chunks, calls itself recursively on these smaller chunks, and combines the outputs into one. We set up conditions which characterize a wide range of divide-and-conquer function definitions. These conditions can be verified by static source code analysis. This way divide-and-conquer functions can be found automatically in existing program texts, and their parallelization based on semi-automatic refactoring can be facilitated. We presents a set of small, semantics-preserving code transformations and a methodology to refactor divide-and-conquer functions in a functional programming language. By applying a sequence of transformations using a refactoring tool, many divide-and-conquer functions can be restructured into a canonical form – which then can be refactored into an instance of a parallel divide-and-conquer pattern. We work out the details in the context of the Erlang programming language.

Keywords: static source code analysis, refactoring, divide-and-conquer algorithm, Erlang

MSC: 68-N15, 68-N19

References

[1] M. Cole, *Bringing Skeletons out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming*, Parallel Comput. 30 (3) (2004) 389–406.

[2] D. Dig, *A Refactoring Approach to Parallelism*, IEEE Softw. 28 (2011) 17–22.

[3] T. Kozsik, M. Tóth, I. Bozó, Z. Horváth, *Static Analysis for Divide-and-Conquer Pattern Discovery*, Computing and Informatics 35 (4) (2016) 764–791.

[4] Z. Mou, P. Hudak, *An algebraic model for divide-and-conquer and its parallelism*, Journal of Supercomputing 2 (3) (1988) 257–278.