Exposing parallelism in sequential code using a modern commutativity analysis

Christos Vasiladiotis
Institute for Computing Systems Architecture
The University of Edinburgh
Edinburgh, UK
c.vasiladiotis@sms.ed.ac.uk

I. Abstract

Extracting and exploiting parallelism is essential to maximizing the performance of legacy code on modern multi-core systems. Traditionally, parallelizing compilers focus on loops, where typically the majority of an application's execution time is spent, and aim to alleviate this task by applying a series of transformations on them. Most commercial and open-source compilers implement basic loop optimizations, but they often fall short when are up against real-world applications. Although advanced loop optimizations have been developed by various research efforts, their adoption has been hindered by lack of supporting analyses or the requirement for the further use of domain expert knowledge (e.g. source code annotations).

In this work, we present a novel parallelization technique using *liveness-based commutativity*, which relaxes the strict data dependence constraints imposed by traditional parallelization approaches. We develop a general parallelization analysis, which aims to extend the scope of automatic parallelization towards more irregular codes. This is achieved by modelling the observable, i.e. live-out effects of computations, as invariants while simultaneously allowing transient computations to violate their data dependence relationships. We develop a set of novel analyses and transformations to verify commutativity, and ultimately parallelizability, for diverse classes of loops.

In our preliminary experiments, we compared our technique on the serial version of the NAS Benchmarks, against a combination of industrial-strength compilers (Intel ICC), a modern Polyhedral compilation framework (LLVM/Polly) and a state-of-the-art research tool that uses idiom recognition. Our experiments revealed that from a total of 1396 loops in the benchmark suite, we were able to identify 1210 loops as commutative, outperforming the joint results of all the aforementioned tools by detecting 576 *additional* loops.

Our immediate goals are to expand our experimental base and explore the potential profitability of the discovered parallelism. In more detail, we aim at evaluating our approach on code bases that use loop traversals of dynamically linked data structures (i.e. pointer chasing), such as the Olden benchmarks and selected programs from the SPEC CPU suites. Our long-term objective is to provide a way to detect parallel algorithmic skeletons as described in our *liveness-based commutativity* framework.