Parallator

Transformation towards C++ parallelism using TBB

Introduction: In the past decades a shift in computer architecture from uniprocessor systems to multiprocessor systems, as multi-core processors, started. It is forced by physical laws that hinder the increase of clock rate further. Instead parallel hardware with increasing number of processor cores is the main source of future performance gains. Parallel programs using multiple threads with independent operations are needed to take advantage of such multi-core processors. A potential for parallelism are «for»-loops where iterations are mostly independent. While compilers might be able to parallelize simple loops with conservative assumptions the potential parallelism can often not be exploited. This lack of expressing parallelism lead to the development of parallel programming models that provide simpler means of expressing parallel executions. Intel's TBB (Threading Building Blocks) library is such a model that provides a high level abstraction to write parallel programs in C++.

Objective: The performance of an application should increase with the number of processing units available. This can be achieved by finding loops and a subsequent transformation to equivalent parallel code that resolves the implied sequential ordering. Finding such loops requires a conscious analysis of program code and a manual transformation may introduce errors. Therefore, we develop an Eclipse CDT (C++ Development Tooling) plug-in that supports the programmer in introducing parallelism in an existing application. As implementation of the parallel code, TBB will be used.

Solution: The resulting CDT plug-in detects iterator- and index-based loops as well as STL «for_each» calls that can be transformed to the equivalent «parallel_for» and «parallel_for_each» counterparts of TBB. While safe fully-automatic transformation is impossible in many interesting cases, we allow the user to agree on a suggested transformation even if it is potentially unsafe to apply it, because of conflicting access to a variable. The implementation also features a natural support for conversion of the loop’s body to C++11 lambdas with capture evaluation, STL or Boost bind functors. The potential code for parallelism is found, using a tree pattern matching algorithms on the program code's abstract syntax tree (AST). As a result, the plug-in allows to introduce explicit parallelism to existing sequential code by replacing loops and STL algorithms with corresponding TBB parallel algorithm calls automatically.