Concept Chapter3 Chapter 3. Concept

Conceptually, SPolly is divided into a speculative loop parallelizer and a non-speculative extension to Polly. Even if the objectives for both parts are the same, namely to improve the performance of loops, the approaches to accomplish them are different. While the former one will introduce speculative parallelism for promising loops at runtime, the latter one tries to weaken the harsh requirements on SCoPs in order to make Pollys loop optimizations applicable on a wider range of loop nests. In the presented setting both approaches may benefit from the polyhedral optimizations and also from parallel execution, so it is hardly surprising that the polyhedral analyses play a decisive role. On the one hand they reveal loop nests which may be optimized by Polly, with or without the extensions of SPolly, on the other hand they are used to detect promising loops to speculate on. Apart from the implementation work, which will be described in the next Chapter, this thesis presents the concepts and key ideas to accomplish both stated goals. We believe that these ideas and the gained knowledge is very valuable not only for future work on SPolly but also for other approaches facing similar situations.

Speculation in the Polytope Model One great benefit of Polly and the underlying polyhedral model is to detect and model loop carried data dependencies. Regarding this fact it sounds plausible to use this ability in a speculative context as well. Unfortunately speculative extensions will compromise this ability because they will assume less dependencies when they weaken the requirements. As less dependencies may lead to more transformations, any speculative extension has to keep track of any change, e.g., to the iteration space, in order to preserve the semantics once the speculation fails. It is beyond the scope of this work to allow arbitrary polyhedral transformations with speculatively removed dependencies, so a more conservative way was chosen and implemented. At first, non-computable memory accesses, as they may arise from aliases or function calls, are overestimated. The resulting polyhedral model is complete with regard to possible data dependencies, but could still reveal e.g., a scheduling for better data-locality or possible vectorization. Afterwards speculation is applied or in other words, the optimized loop nest is speculatively parallelized. Note here, that Polly can only change the scheduling based on the overestimated and therefore complete polyhedral representation of the loop nest, thus only in a sound way. A more detailed description on both steps is given in the following Sections, especially SpeculativeParallelExecution, SpeculationFreeOptimizations and OverestimatingDependencies.

Speculative Static Control Parts sSCoPs As Polly is restricted to static control parts, there is a counterpart limiting the applicability of SPolly. This counterpart is called speculative static control part, or short sSCoP. Both are the central entities of the respective tool and they only differ in the restrictions on the underlying region. In contrast to Polly, SPolly allows arbitrary function calls, possible aliases and non-affine base pointers of memory accesses.

To illustrate the differences, Figure fig:WeakendSCoPs presents three sSCoPs where each one validates one of the weakened restrictions, thus is a sSCoP but no SCoP. As a comparison to the restrictions on SCoPs (Table tab:SCoPRestrictions), a full list of sSCoP restrictions is given in Table tab:sSCoPRestrictions on page sSCoPRestrictionsPage.

frame=none figure[htbp] *-5mm [Speculative static control part violating the function call restriction on SCoPs] minipage[c][3.5cm]0.3 tabularc Primitives/Code/sSCoP1.c