==========

Summaries for the second meeting of the seminar "Automatic Parallelization in the Polyhedral Model"
Topic: "Introduction to Polyhedral Program Optimization"

==========

Summary 1 of 4

The second chapter of the PhD-thesis by Pouchet introduces the polyhedral model as a mathematical framework for loop-nest optimization. The polyhedral model differs from classic representations in that it does not model the inductive semantics of programs, but takes the more fine-grained approach of representing the instantiations of statements w.r.t. loop-variables. This makes it possible to perform and validate program transformations done at the level of statement schedules.

Polyhedral program optimization is a three-part process: First, the input program is analyzed and its polyhedral representation is constructed. Second, the polyhedron representation is optimized by picking a transformation strategy. It is notable that this transformation is done in a single step; i.e. there is no sequence of transformations as with usual optimization schemes. This transformation mostly consists of a so-called affine schedule, which is complemented by the iteration domain and array subscripts. Finally, code is generated from the polyhedral representation by „scanning“ the polyhedral in an order imposed by the schedule.

The process of constructing the polyhedral representation imposes a strict form on the input programs, restricting them to for-loops and if branches which only use affine forms of iterations variables for conditions and statements amongst other things.

A polyhedral statement is defined as any single statement which does not affect the control flow of a program and uses only affine subscripts in its expression (e.g. an assignment to/from an array). Furthermore the instantiations of any such statements are captured in the possible values of the surrounding loop-iterators in the form of a set of affine inequalities. The iteration domain is then the set of lattice points enclosed by these (possibly parameterized) inequalities, forming a polyhedron. Finally, memory accesses through arrays are encoded in the form of access functions, which describes how each read reference accesses memory in terms of surrounding iteration variables and global parameters.

The goal of the program transformation is to infer a so-called schedule. Given any statement, a schedule will describe when each instantiation of this statement is executed. A schedule, if multi- or one-dimensional, consists of an affine form of loop iterators of surrounding loops and global parameters. Each point of the iteration domain can then be assigned a timestamp using the schedule. The lexicographic ordering of these timestamps then corresponds to an abstract execution order. Of particular interest are transformations which do not violate dependencies between statements. For this, dependencies are modeled by another polyhedron for any dependence between two statements which then describes if the dependence exists for any particular instance of iteration vectors. A transformation is the said to be legal iff the lexicographic ordering preserves the order of the dependencies over the polyhedron.

Finally, the code generation phase generates a scanning code over the iteration domains in the order imposed by the schedule. How exactly the code generation works is postponed to later chapters, but the author notes that a different schedule will always produce different syntactic code, even if the schedules lead to the same order of instances.

==========

Summary 2 of 4

PhD Thesis by L. N. Pouchet, Chapter 2

Beginning by the mathematical representation of polyhedra and the source to polyhedral transforma- tion, everything up to final code generation is shortly introduced in this Chapter of Pouchets PhD thesis. Furthermore, it includes the correspondence between different kinds of scheduling (one- and multidimensional) and classical, well-known loop transformations. In various examples representation, transformations and code generation are illustrated and different approaches in research and industry are shortly described and partially compared. Before a short paragraph about syntactic code generation completes the Chapter, the legality of (scheduling) transformations is discussed. Two approaches to enforce them are stated; one which encompass the legality criterion directly into the search space and one with posteriori checks and corrects the scheduling if needed.

PhD Thesis by C. Bastoul Chapter 2 & 3

As Pouchet, Bastoul first introduces the concepts behind the polyhedral model, namely static control parts (SCoPs) together with their building blocks: static control loops, static control conditionals and affine array accesses (called static references). An algorithm to detect them is given as well as approaches to increase their number in real-life programs. He also evaluates effectiveness of the algorithm on several benchmarks and presents additional information on the discovered SCoPs.

In the second Chapter scheduling functions are introduced, first to represent program execution and afterwards as transformations in the polyhedral model. The Chapter ends, similar to Pouchets introduction, with a discussion about the legality of such (scheduling) transformations.

Bastoul first states that for two statements the first s elements of their iteration vectors (where s is the number of shared loops) combined with the textual order of the statements is enough to order them, thus to create a scheduling function for each statement. Then he describes why the set of scheduling functions is often restricted to affine ones (which agree on this order) to ease the legality check and the code generation. He also presents a simple algorithm to construct such affine functions by interleaving the AST with the textual-ordering of statements.

The second part of this Chapter introduces a new approach to apply polyhedral transformations. Instead of changing the basis of the source polyhedron additional dimensions are added. Those dimen- sions increase the complexity for later steps but also allow arbitrary transformation functions, especially non-affine, non-uniform and rational ones. He states that previous work, which removed or weakened some of the restrictions on scheduling functions (thus allowed e.g., non-affine ones), is completely coped by his model. To verify this statement he interprets the auxiliary dimensions for each restriction and describes why the ordering of statements is preserved.

The last part of this Chapter deals with the legality of transformations. Utilizing the dependency types described by Allen and Kennedy legal transformations are exactly those which do not modify the order of dependent statements. An affine way to represent (data) dependencies in a SCoP is given and used to restrict the transformation search space to legal ones. Furthermore, Bastoul describes a legality check as well as possible corrections for non legal transformations.

Questions:

1. A wider range of scheduling functions yields the need for better heuristics as more than one optimization goal is possible (spatial- and time-locality, parallelism, . . .). In this context questions similar to the following ones might arise:
   • How to choose on parameters (e.g., for loop tiling)?
   • Prefer small SCoPs (e.g., enforce loop fission) or large ones (e.g., apply loop fusion if possible).

2. Extending the first question; which (system dependent) information is useful (and why) to predict the effect of different loop

transformations?

3. While some problems introduced by real-life programs are discussed, aliasing and function calls are still without (satisfactory) solution. Which techniques or analyses might be useful to avoid inlining (not efficient regarding both time and space) and programmer annotations (special knowledge needed) for those cases.

==========

Summary 3 of 4

To perform program transformations we need to first agree on a program model. We've to find a good tradeoff between representativeness and analysis power. The well known static control program class is probably a good candidate. This class only allows do loops, if conditionals with affine bounds / conditions and arrays which only use outer loop counters, structure and size parameters. The iteration space of a loop can be characterized as a polyhedron which is a convex set of points in a lattice which can be described by affine inequalities.

A useful technique before optimization is preprocessing. There is for example constant propagation, while to do loop conversion, goto elimination, loop normalization, inlining to allow the program to fit in the SCoP class. The static control parts can then be generate by a simple traversal of the syntax tree.

The advantage of this choice is that on average most time in a program is spent on few loops and the volume of accessed data (in loops) is potentially very high.

Many transformation techniques have been studied independently but the problem is that each has its own properties which means legality test or cost model. But how to decide which transformation to use after all? The polyhedral model allows us to handle multiple transformations in one framework. The framework uses a scheduling function which defines the execution order. Transformations are therefore described as affine scheduling functions.

To test if a transformation is legal we can create dependence polyhedras. Then we can extract the legal transformation space constraints and finally check if a transformation is valid or not. With this method we can also use a transformation with parameters and try to find a legal transformation in the legal transformation space.

Finally the most important thing: Code Generation. We need to generate fast code given the scheduling function. The code generation stage generates a scanning code of the iteration domains of each statement with the lexicographic order imposed by the schedule. Statement which share the same timestamp are most normally executed under the same loop (fusion). The generated code is typically AST-based and can be translated to some language. To achieve better results one can translate this back to polyhedral and refine the schedule until some criteria are met e.g. code size or control complexity. For many years this stage was the bottleneck only allowing certain schedules but this has been "fixed".

Open questions:
1) How to generate the "scanning code"?
2) What to do to ensure that the generated code is really faster and not much slower?
3) How does the Farkas Lemma and the dependence polyhedra interact?

==========

Summary 4 of 4

Summary:Polyhedral Model
Representing a program (in syntaxt ree, call tree, control-flow graph, SSA) way is not convenient for good optimizations as, due to compilation-time constraints and to the lack of an adequate algebraic representation of the semantics of loop nests, traditional (non-iterative) compilers are unable to adapt the schedule of statement instances of a program to best exploit the architecture resources.

To build complex loop transformations, an alternative is to represent programs in the polyhedral model. It is a flexible and expressive representation for loop nests with statically predictable control flow. A polytope model is described as An affine hyperplane is an m−1 dimensional affine sub-space of an m dimensional space.Here we divide control and data structure in order to handle as many operatins SCops, which is consecutive statemts in the program with convex polyhedral iteration domain,so that instancewise dependence analysis is possible.

Polyhedral model has two types of transformations (1)Scedule only and (2)Schedule and Representaions.These transformation mofify the polyhedra containing same points into target polyhedra but in a new coordinate space.A central concept of program optimizations to preserve the original semantics of the program since all transformations are not affine so we have to check for the legality of the transformations.A transformation is said to be legal if it does not modify the order of dependent pair of the statements or if it preserves the temporal sequence of all dependencies.

Polyhedral transformations experimentally showed that they require only reasonal amount of time and resources but also gives us an idea scalability required for worst case expontential algos.However polyhedral model only handles unimodular and at least vertibile functions

Questions
(1)How to compute $Ax + a \geq 0$ with the help of given code? lot of example are given but I am not sure that how these matrices are computed.

(2)What is the use of SCops extraction? how rich SCops are useful.

(3)Why output dependemcies , antidependencies are termed as False Dependencies?

==========