==

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 Omega Test

The Omega Test can determine whether a dependence exists between two array references and under what conditions. It uses integer programming to produce the results which is considered too expensive. But the Omega Test is competitive with approximate algorithms used in practice and could be used in production compilers.

To check if two array references have a dependence we first need to extract equalities and inequalities from the given program with respect to loop bounds. On top of that the Omega Test combines new methods for eliminating equality constraints with an extension of Fourier-Motzkin variable elimination. Apart from that there are other implementation details which give a speed improvement.

The first step is to normalize all constraints and scale all constraints which don't have integer coefficients. Then we start with the equality constraints by eliminating one after an other. If an equality has a coefficient of 1/-1 we can directly solve the equality and substitute the result into all other constraints. If not we use a technique which introduces a new variable and shrinks the coefficients. By repeated application we get a unit coefficient and can eliminate that constraint. The next step is to handle inequality constraints. We first check for direct contradictions, tight inequalities and redundant inequalities and report failure, create an equality or eliminate those respectively. If we are done with that we try to disprove the existence of integer solutions by creating a new problem that contains all inequalities that do not involve a specific variable and all inequalities produced by combining each par of upper bound and lower bound on this variable (see Fourier-Motzkin). The difference to Fourier-Motzkin is that we create a second new problem and if the integer points described by both new problems are the same there is an integer solution to the original problem. Otherwise we need to make a check on every lower bound in turn. When performing exact transformations we try to choose the variable that minimizes the number of constraints while for non-exact reductions we choose a variable with coefficients as close to zero as possible.

The actual implementation uses hash keys and constraint keys to allow faster search for redundant and opposing constraints. After normalization if we found no multi-variable constraints, we know the system must have solutions. The Omega Test simply decides if there is a solution to an integer programming problem but with some modifications it is possible to allow it to be used for simplification. The main point is to never perform Fourier-Motzkin variable elimination on protected variables and some other techniques.

Open question:
1) Is this used in modern compilers? If not, why?

==

Summary 2 of 4

Omega Test

Omega test is used to dependency analysis, Does a dependence exist between two array references? and under what conditions?. Omega test is the extension of Fourier Motzkin variable elimination method. Here we consider the analogy of integer programming problem. The test following steps (1)Generate the set of constraints (2)Normalize and tighten the constraints (3)eliminate equality constraints (4) eliminate inequality constraints and in last use Fourier Motzkin method to detect the real and integer solution.

The omega test nightmare is that it takes time proportional to the absolute value of the coefficients. so the paper present the nice implementation that only deal with integer ; no rational, to improve the performance. It is implemented via following steps (1.)use Fourier Motzkin elimination on unbounded variables simply deletes all constraints involving it. (2)As constraints are normalized, enter them into a hash table based on their constraint key and this allows us to check for redundant, contradictory, or tight constraint pairs in constant time per constraint. (3)After normalization, if no multi-variable constraints exist, the system must have solutions. Return immediately. (4) then Fourier Motzkin variable elimination is applied.

The Omega Test can handle nonlinear subscripts such as integer division and integer remainder operations and Use of projection enables the calculation of dependence distance and direction vectors efficiently. Projection can

an also be used when compile-time analysis is not possible and can produce a predicate that allows run-time determination of whether a particular dependence or dependence direction exists.

Performance wise even though the integer programming problems are NP-complete, omega test performs reasonable for almost all the programs that are in practice. Normalizing the constraints and checking for contradictory or redundant constraints require O(mn) time as hashing is used also subproblem generated from Fourier Motzkin method takes time proportional to size of the subproblem.

Questions

1. why it is more easy to deal with the equality constraints?
2. Is it possible that Fourier Motzkin results in too many sub-problems that eventually degrade the performance?

3. Significance of protected variables when we already having solution in P?

==

Summary 3 of 4

The Omega Test: a fast and practical integer programming algorithm for dependence analysis
William Pugh describes in his work the Omega Test, a new integer programming approach for exact de- pendencies testing. He states the runtime is competitive with approximate algorithms used in productive compilers but additionally results, e.g., dependence direction, can be achieved by slight modifications and with adequate overhead (at least for the benchmarks tested).

The basic idea is to test if an integer programming problem computed for each pair of array sub- scripts in a loop nest has a (integer) solution. First, simple conditions for non-feasible equalities and contradictory inequalities are tested before a Fourier-Motzkin approach is used to reduce the number of variables. In the basic framework only a decision problem is given (namely if the subscripts are depen- dent or not), thus it is sufficient to find one infeasible constraint. To lower the runtime in regular cases a lot of special cases (early exits) are implemented which may provide an answer without eliminating all variables.

Additionally results (e.g., distance vectors) can be achieved by a modified version of the Omega Test. Two ways are described, one which will yield the dependence direction and distance vector of two subscripts and one which offers even more information on the dependency.

In the end the approach is compared to other approximate and exact techniques. Several of them utilizing integer programming some even Fourier-Motzkin elimination but most of them state an addi- tional overhead compared to specialized or approximate polynomial solutions. A theoretical comparison is given to counter this statement for the special cases other solutions provide exact results. Further- more, an evaluation on different benchmarks, which shows the analyses time compared to the copy time of the problem, is shown. The results should proof the applicability in productive compilers.

While the usefulness of exact dependency results and additional information is clear, it remains unclear how the Omega Test performs compared to other techniques and why copy time is an appropriate reference.

Open/Further Questions

- The described hashing technique and the purpose is quite a bit confusing, so how and why is hashing used?

- Why is the analyses time compared to copy time and why is loop bound/ subscript scanning so expensive?

- As stated, some non-linear constraints, e.g., max or min values, can be handled but questions might arise as:
  – How applicable is this to general purpose code?
  – What extensions could allow more general subscripts (quadratic solvers)?

– How could more expensive approaches be used in modern (high performance) machines to

handle more cases with acceptable runtime?

• While polynomial complexity of the Omega Test is comparable to approximate algorithms (in special cases), why is neither runtime, nor precision (for non-exact results) is evaluated?

==

Summary 4 of 4

Omega test is a integer programming algorithm which is used to do the analysis of array dependence like Does a dependence exist between two array references? and under what conditions? The algorithm present the evidence that integer programming is no so expensive to be used for dependence analysis, which is competitive with approximate method in practice and applicable in production compliers. Omega test is the extension of Fourier Motzkin variable elimination method which is NP-complete generally having exponential complexity in worst but polynomial in most situations.

The basic observation is that data dependecy problems can be equivalent to deciding if there is a integer solution to a set of linear equlities and inequlities as integer programming. Omega Test combines new methods for eliminating equality constraints with an extension of Fourier-Motzkin variable elimination to integer programming. The rough idea is to reduce the problem into a simpler domain which have fewer variables. This simplication can be done by projecting 'n' variables to 'n-1' variables. At same time, the solution of simplified domain can imply the solution to the original problem. The first step is to Normalizing (and tightening) constraints which will scale the rational coefficients to produce integer coefficients. The method is to divide all the cofficients by common di- visor of the cofficients. The final normalized constraint in one in which all the coefficients are integers and the greatest common divisor of the coefficients is 1. Afterwards, we starting eliminate all the equality constrains to produce a problem in inequalities. Each equaliti elimination will eliminates a variable. If the variable's absolute value is 1, we just solve for this variable and substitute in system. Otheriwse we have to replace the equation with another constrains(see paper 2.2). In the 3rd we dealing with inequalities. The deitails is like following, if we can find the contradicting inequalities then it report no solution. Other- wise, we remove inequlities by tighter ones and remove reduandant constrains, if problem has only a sungle variable then we report a solution else we continue eliminating one variable and recurse.

Implementation details of Omega test needs to represent the equalities and inequalities as vectors of coefficients which is only in interger formal. Perform- ing Fourier- Motzkin elimination on an unbounded variable which has no lower bounds or upper bounds, simply deletes all the constraints involving it. We re- cusively delete the unbounded variables. Next, we normalize all the constraints and assign hash keys and constraint keys to them. hash keys and constraint keys can lead faster search for redundant and opposing constraints. Finally, we examine the variables to decide which variable to eliminate.

The time to eliminate one equlity is O(mnlog|C|) in worst-case, where C is the coefficient with the largest absolute value. Eliminating unbound variable takes O(mnp) worst-case time, where p is the number of passes reuqired to elim- inate all the variables that become unbound. And normalizing the constrains and checking for directly contradictory or redundant constrains needs O(mn) expected time. And producing the subproblems resulting from Fourier- Motzkin variable elimination takes time proportional to the size of the subproblems pro- duced.

Questions

• How to handle symbolic constants by programming methods?

• what are the limitations and advantages?

• How does this Hash work for normalization exactly?

==