Project:-

Abstract: - To develop an exact test to determine data dependence in statements in an arbitrarily nested loop with constant bounds using the Karmarkar’s algorithm and integer programming. Such tests are vital to determining whether a particular loop of a program is parallizable or not. This test can be used in an automatically parallizable compiler. This test developed is an exact test which means it tells exactly whether the loop is parallizable or not without any assumptions on the bounds of the loop (except the fact that they are constant).

Introduction:-

As the limit to sequential computing is reached, processors are going parallel. So are the programs written for them. Key to this process are the automatic parallizable compilers. Such compilers typically analyze the problem of data dependence. Many different data dependence tests exist like the famous gcd and banerjee tests which are incredibly efficient but they are in-exact (do not consider bounds). Many exact tests also exist (eg- omega and LaPuTa tests) but they are inefficient on large inputs. Often in the process of data dependence analysis an optimization algorithm is needed (linear programming methods like Simplex method, ellipsoid method etc). Almost all the methods use the simplex algorithm but its theoretical worst case (O(2^n)) means it is inefficient for large inputs and thus other methods are needed, especially polynomial time methods. Particularly useful is the interior point method known as the Karmarkar’s algorithm, invented by Narendra Karmarkar in 1984. Under certain circumstances (independent of input) its worst case is always polynomial time in nature. This paper tries to implement the Karmarkar’s algorithm and use it ultimately to solve a data dependence problem. This paper also discusses a few other algorithms and ideas to increase efficiency. This paper also discusses the results of the implementation of some sample problems and their profiling data obtained. Finally this paper discusses the possible use of this test in real life parallel computing and the limitations of this method.

Data Dependence:-

Consider any two statements S1 and S2 in a program. A data dependence exists between the two statements if the input of one statement directly affects the output of the other statement.

The data dependence statements that we are interested in are between statements inside a loop of arbitrarily nested loop.

**The types of data dependence are:-**

True dependence:- A true dependency, also known as a data dependency, occurs when an instruction depends on the result of a previous instruction:

Consider two statements S1 and S2. S1 depends on S2 if:

O(S1) ∩ I (S2) , S1-> S2 and S1 writes something read by S2

Anti dependence:-An anti-dependency occurs when an instruction requires a value that is later updated

Consider two statements S1 and S2. S1 depends on S2 if:

I(S1) ∩ O(S2) , mirror relationship of true dependence

Output dependence:- An output dependency occurs when the ordering of instructions will affect the final output value of a variable

Consider two statements S1 and S2. S1 depends on S2 if:

O(S1) ∩ O(S2), S1->S2 and both write the same memory location.

Types of data dependence particularly useful to analyze in loops:-

Loop independent dependence:- If the data dependence exists between statements within a particular iteration of a loop, then there exists a loop independent dependence.

Loop carried dependence:- If a dependence exists between statements in distinctly different iterations in a loop, then it is called a loop carried dependence.

Solving a data dependence problem is achieved by solving an equivalent linear Diophantine system, whose description is given below. In other words we are modeling a data dependence problem into an equivalent integer programming problem.

Linear Diophantine systems:-

Consider the linear system AX=B where A is an m\*n matrix, B is an m dimensional column matrix and X consists of all the n variables which comprise the system. If {x1,x2,…xn} =X and if all the x’s in this column matrix can only possess integral values, then this system is called a linear Diophantine system of equations.

A problem involving linear Diophantine systems and obtaining its solutions is called integer programming. Solving a data dependence problem is equivalent to determining whether there exists an integer vector x to a linear Diophantine system that corresponds to the loop in question.

Eg- Consider loop

for i=1 to 10

for j=1 to 5

A[i]=B[j+1]

Is equivalent to solving the linear Diophantine system:-

x1-x2-1=0 ; 1<=x1<=10 and 1<=x2<=5

Typically if there exists an integer solution to this equivalent dipohantine set then there exists a data dependence in the loop. Also to establish data dependence there must be atleast two equations involved.

We have solved this problem in a very simple brute force method. The number of iterations in the worst case scenario = (the sigma term). The worst case corresponds to there does not exist any integral solution to the system. The problem is always finite provided there exists finite constant bounds for each of the variables in the system.

Iteration vector:-

For a loop, an iteration vector is a member of the Cartesian product of the bounds for the loop variables

The Integer Programming Algorithm:-

The algorithm used in determining solutions is remarkably simple. The problem is n dimensional.

1. Define a column vector X={x1,x2,x3…} (iteration vector) , for each li<=xi<=ui
2. Assign X to its lowest possible value (each of xi=li)
3. Define Totalno=0
4. For i=1 to n do

4.1. Totalno =Totalno + (u[xi]-l[xi]+1)

1. Declare tracker=n
2. For I=1 to totalno do:-
   1. . Check if AX=B
   2. .If true and l[tracker]<=X[tracker]<=u[tracker]then halt program, declare X as solution and exit
   3. Else X[tracker]++
3. If X[tracker]>u[tracker] or X[tracker]<l[tracker] do:-
   1. tracker- - until l[tracker]<=X[tracker]<=u[tracker]
   2. Call reset function.

The reset function does the following operations:-

1. x[tracker]++
2. Reassigns all x[i] to l[i] where i ranges from tracker to n-1
3. Re-declare tracker=n

Eg - If the problem is 0<=x1<=1 and 0<=x2<=1 then the possible integer solutions checked are:- (0,0),(0,1),(1,0),(1,1).

Efficiency:-

The algorithm is especially efficient if the number of variables is less and the bounds are small. It has a polynomial time complexity O(mod(nk)) where k is an integer but k can be very large depending upon the input. In order to increase efficiency of the problem, the following can be done:-

1. Reduce the problem to a row reduced echelon form.(View Appendix)
2. Use the karmarkar’s algorithm to narrow down the search (reduce the bound lengths for each of the variables).

How is the Karmarkar’s algorithm and the integer programming algorithm combined:-

Technically the Karmarkar’s algorithm has no correlation with a linear Diophantine system of equations. The Karmarkar’s algorithm is typically an optimization algorithm than minimizes an objective function ( a polynomial in n dimensions), subject to a set of constraints(a set of equations). In order to reduce the bounds repeatedly apply the Karmarkar’s algorithm with a predefined set level of precision with objective function z=+-xi where xi is each of the variables in the system. This gives the upper and lower bounds of the variables under the constraints of the system. Then we assign each of the new bounds as the follows:-

(New l[xi],New u[xi])=(Ceil(max(Old l[xi],Minfound[xi])),Floor(min(Old u[xi],Maxfound[xi])

Where Maxfound[xi]=z=-xi upon application of the Karmarkar’s algorithm

And Minfound[xi]=z=+xi upon application of the Karmarkar’s algorithm.

Now after all the reduced bounds under constraints are found, we run the integer programming algorithm to find whether an integer solution exists or not and thereby conclude whether data dependence exists or not.

Limitations:-

The limitations faced by this method as are follows:-

1. This method is not very efficient to small inputs (2 or 3 variables and less than 5 equations) as compared to the simplex method.
2. This method cannot solve the problem for variable loop bounds.
3. The input in the Karmarkar’s algorithm does not actually give correct output if the minima of the objective function are unbounded or unfeasible. (Unless the termination criteria and the Karmarkar’s modified f function’s definitions are redefined).
4. The accuracy of the bounds obtained after Karmarkar’s algorithm is implemented may be in fact greater than the original bounds, (especially if the sum of all the upper bounds is a relatively large number).
5. Typically this particular algorithm is made for loop bound increments of one (we can modify the algorithm accordingly to accommodate any constant loop bound increment).
6. This technique is inefficient in case no dependence exists.

Profiling:-

Profiling techniques used:-

To calculate the number of times the principal loop of an algorithm is run, we have used gprof program on Linux (Suse) operating system. To find the time required for execution of the program excluding I/O we used gettimeofday( ) function in sys.h and created user defined functions using c library structures to get time difference at microsecond level at beginning and end of execution and took their difference to calculate time taken. To calculate memory required and time taken to compile we used Netbeans IDE. The computer used for profiling is an 8 core intel xeon processor, each of core an 8 GHz processor.

For the Karmarkar’s algorithm:-

Examples examined:-

Minimize x1-3x2 +3x3

Subject to:- x1 -3x2 +2x3=0

x1+x2+x3=1

{x1,x2,x3}>=0

Data obtained:-

Number of times principal loop called=16

Time taken for execution= 0.383 ms

Time taken for compilation= 201 ms (this was done on a different platform)

Answer obtained= {0.748,0.252,0} {Rounded figure}

Theoretical answer={0.75,0.25,0}

For the integer programming algorithm:-

Examples examined:-

1. x1+x2=1

Subject to 1<=x1<=10 and 1<=x2<=10

Data obtained:-

Number of times main loop ran= 100

Time taken for execution= 0.247ms

Time taken for compilation= 107ms

Answer obtained=Dependence does not exist

Theoretical answer=Dependence does not exist

1. x1+x2=3

Subject to 1<=x1<=3 and 1<=x2<=4

Data obtained:-

Number of times main loop ran= 2

Time taken for execution= 0.09ms

Time taken for compilation= 108ms

Answer obtained=Dependence does exist, it exists at (1,2)

Theoretical answer=Dependence does exist

Appendix:-

Appendix (A): More facts about data dependence:-

Consider two statements S1 and S2.To classify data dependence, compilers use two important vectors: the **distance vector** (σ), which indicates the distance between fn and hn, and the **direction vector** (ρ), which indicates the corresponding direction, basically the sign of the distance.(Here fn and hn are coordinates or integer values possessed by the two statements S1 and S2 respectively).

Dependence direction:-

The **distance vector** is defined as σ = (σ1 , ... , σk) where σn is σn = in - jn

Dependence Distance:-

The **direction vector** is defined as ρ = (ρ1 , ... , ρk) where ρn is:

* (<) if σn > 0 => [fn < hn]
* (=) if σn = 0 => [fn = hn]
* (>) if σn < 0 => [fn > hn]

Various other criteria for classifying dependence:-

A dependence between two operations: *a* and *b*, can be classified according to the following criteria:

* **Chronological order**
  + If *Sa* < *Sb*, this is a lexically forward dependence
  + If *Sa* = *Sb*, this is a self-dependence
  + If *Sa* > *Sb*, this is a lexically backward dependence
* **Loop dependence**
  + If all distances (σ) are zero (same place in memory), this is loop independent
  + If at least one distance is non-zero, this is a loop carried dependence

Appendix (B): Reduced Row echelon form:-

The reduced row echelon form is a form of a matrix which is attained by a finite number of row or column operations which follows the following rules:-

1. All nonzero rows precede the (that is appear above) zero rows when both types are contained in the matrix.
2. The leftmost nonzero element of each nonzero row is unity (the number 1).
3. When the last nonzero element of a row appears in column c, then all other elements in column c are zero.
4. The leftmost nonzero element of any nonzero row appears in a later column (further to the right) than the leftmost nonzero element of any preceding row.

The following theorem allows a particular approach to increase the integer programming efficiency:-

Theorem:-

To solve the system of linear diophantine equations AX=B,unimolar row reduce[ At|I] to [R|T],where R is in row echelon form. Then the system AX=B has integer solutions if and only if the system RtK=B has integer solutions for K and all the solutions of AX=B are of the form X=TtK.

This can be used to make a more efficient test that solves a reduced problem with fewer steps. Reduced row echelon form are special cases of row echelon forms.

The drawback to this method is that one cannot use it to have a guaranteed reduction in effort. But in most cases there is a reduction is computation time and generally there is a reduction in effort.

Appendix (C): SLDS-2V algorithm:-

A particular algorithm was analyzed with respect to data dependence analysis. This algorithm is applicable to two variable systems and is more efficient than the integer programming algorithm. The original algorithm actually finds out all the solutions to the system and thereby could be used to find out dependence distance and direction etc, but for our application we terminated execution of the algorithm’s implementation as soon as we found out an integer solution to the problem. The algorithm is based on the extended gcd algorithm (a modification of Euclid’s gcd algorithm).

Extended gcd(greatest common divisor) algorithm:- (This algorithm is also used in the gcd test).

1. Define a recursive function gcd(a,b) which defines gcd of a and b.

2. Let a=b\*q + r here q is the quotient and r is the remainder

3. Recursively define gcd(a,b)= gcd(b,r) for r!=0 {the pure gcd algorithm}

4. When r=0 declare the last b obtained as gcd(a,b).

The actual extended gcd algorithm is implemented differently. (Refer reference 2 page 184)

The SLDS-2V algorithm:-

The SLDS-2V algorithm actually calculates the solutions to the problem(the integral solutions). We cut it short while implementing it to exit after a solution has been detected. The exact algorithm uses the extended gcd algorithm to calculate gcds of the constraints and uses a series of operations to find the solutions.{ For the exact algorithm refer reference 2 page 187}

It is actually more useful to use this algorithm in case of 2 variable systems rather than the algorithm developed in this paper.

References:-

1. http://www.math.uwaterloo.ca/~wgilbert/Research/GilbertPathria.pdf{Linear diophantine equations William J Gilberth and Anu Pathria}

2.Algorithmic number theory: 7th international symposium, ANTS-VII, Berlin ...

By Florian Hess, Sebastian Pauli, M. Pohst

3.Linear Programming by Howard Karloff

4.Solving a system of diophantine equations with

lower and upper bounds on the variables

Karen Aardal Cor Hurkensy Arjen K. Lenstraz

5.COMPARATIVE ANALYSIS OF THE AFFINE SCALING AND KARMARKARâ€™S

POLYNOMIAL â€“ TIME FOR LINEAR PROGRAMMING

ADEJO B & CHOJI D.N.

6.Optimising compilers for modern architectures-A Dependence-based Approach by Randy Allen and Ken Kennedy

7.The power test for data dependence- Micheal Wolfe,Member IEEE and Chau-Wen Tseng