Partitioning of Loops with Non-Uniform Distance Data Dependence for Parallelization

Archana Kale* Amitkumar Patil†
Supratim Biswas‡
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

Abstract

The extent of parallelization of a loop is largely determined by the dependences between its statements. Loops with loop carried, uniform(constant) distance are easy where as, loops with non-uniform(variable) distance are considerably difficult to partition and hence, parallelize.

Contemporary researchers have used, unimodular transforms, chain of recurrences, polyhedral model, iteration slicing etc for partitioning iteration space with non-uniform dependences. The distinctiveness of our approach is in targeting Linear Diophantine Equations (LDEs), which capture dependence. Examining solutions of LDEs, we have identified a) interesting patterns among dependent iterations and b) a generating subset of dependent iterations. These patterns were observed in parametric solutions of 2 variable LDE and subsequently generalized.

We define a group of all interdependent iterations as a component(C) and our partition(P) is a collection of independent components. This dependence represented is termed as precise because it accounts for all the iterations and their dependences accurately. We give expressions for bounds on the size and number of components and also generator iteration or seed of a component.

Our generalized characterization of LDEs covers a wide range of commonly found array subscripts. We solve a multi variable LDE by converting it to a set S of 2 variable LDEs. We show that solutions of a small subset of S can generate solutions of S. We build partition of multi variable LDE by nontrivially merging partitions of S. Finally our approach is extended to a system of LDEs. Results of limited experimentation show that our partition exploits sparsity and large number of independent iterations, usually present is iteration space with non-uniform dependences, to obtain significant iteration level parallelism.

Category: D.3.4 Code Generation, Compilers, Optimization

General Terms: Dependence Distance, Non-Uniform Dependences, Iteration Space Partition, Parametric Solutions, Linear Diophantine Equation, system of LDEs, Significant Parallelization

1 Introduction

Loops with loop carried constant(uniform) distance data dependence(CD) having cycle free Data Dependence Graph(DDG) have been successfully dealt with in the literature[6]. A few researchers have also considered loops having cycles in DDG[2, 3]. Some efforts have considered loops with variable(non-uniform) distance data dependence(VD) and DDGs with or without cycles[3, 4, 5, 7, 8, 9, 11]. Unlike the case of CD, the solutions of LDEs for VD do not seem to have a regular structure among dependent iterations.

Our approach builds on a two variable LDE for which parametric solutions are well known [1, 10]. We have analyzed the solutions and have developed a mathematical formulation that captures the structure among dependent iterations. The structure leads to partitioning of the iteration space into components. Each component of the partition represents iterations which have to be executed sequentially and distinct components are parallelizable. Further, bounds on the number and size of components have been obtained. Examining the structure of a component, we show that all iterations of

*archanak@cse.iitb.ac.in
†amitkumar.bvb@gmail.com
‡sb@cse.iitb.ac.in
Paper for PACT 2015
Draft—Do not Distribute

the component can be generated from any single, representative iteration, called its seed. The representation of the partition is consequently reduced to a set of seeds. This set can be used as a basis to generate and schedule the components dynamically according to the demands of the environment.
We have extended this approach to partition the iteration space for multiple LDEs by combining the components of partitions of the individual LDEs. The correctness of the non-trivial composition of partitions to form the resultant partition is proved in step 5 of section 5.1.
The applicability of our approach rests on its ability to extract exploitable parallelism from VD loops with multiple LDEs. Experimental evaluation of effectiveness of our approach requires existence of VD loops with large number of LDEs. We have compared our results with some examples present in literature[5, 7, 9]. Additionally, we have created multiple LDEs with random coefficients for further experimentation. Results show that loops with VD offer reasonable parallelism which reduces as the number of LDEs increase. To increase the parallelism in case of large number of LDEs we have formed an alternate partition using a heuristic. This heuristic shows significant improvement in parallelism.
The paper is organized as follows. Motivation is in Section 2, Section 3 presents the basic mathematical formulation for a 2 variable LDE. Section 4 presents various LDEs representing different types of VD loops. Sections 5 and 6 extend the basic solutions to all types of LDEs. Comparative Analysis, Experimentation and Conclusion are presented in Sections 7, 8 and 9 respectively.

2 Motivation

The approach is motivated by the following example of a VD loop.

```
for(i = -8 ; i < 8 ; i++)
{ S1: g[2i + 1] = .........;
  S2: ......... = g[3i + 6]; }
```

The dependent iteration pairs of the loop are modeled by LDE, $2x - 3y = 5$. The solutions of the LDE, { (-8,-7) , (-5,-5) , (-2,-3) , (1,-1) , (4,1) , (7,3) }, over the range R=[-8,7], give all the dependent iteration pairs. This leads to a partition P over R, as shown in Figure 1, where iterations are nodes and edges represent dependence between them. Each Component(C) is a set of dependent iterations which should be executed sequentially and partition $P = \{C_i\}$. Different Cs can be executed parallely without synchronization.

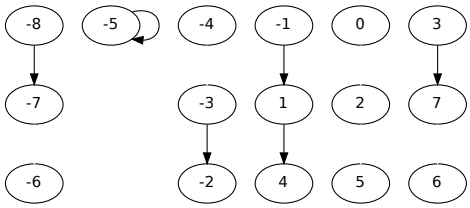

Figure 1: Partition of [-8,7]

Partitioning the iteration space exposes all the parallelism naturally present in the loop. Here $|P| = 11$ is the maximum number of parallely executable Cs and the size of largest C, $\max(|C_i|)$ is 3. Among many possible semantically equivalent schedules consistent with the partition of Figure 1, a realizable, load balanced, 4-thread parallel schedule for the loop is:
{{-8,-7,-6,-5},{-3,-2,3,7},{-4,-1,1,4},{0,2,5,6}}.
This example though simple, brings out the benefits of Parallelizing a VD loop.

Our approach rests on using solutions of LDE of the form $ax + by = c$ where, $a > 0$ and $a \leq |b|$. This is the normalized form used in the paper without loss of generality.
Given an iteration space and a set of LDEs, we use the term **Precise Dependence(PD)** to denote the collection of all solutions of all LDEs in the iteration space. The LDE characterizes VD except for the case $a = 1$ and $b = -1$ where it characterizes CD. Iterations which do not appear in PD are independent iterations.

3 Two variable LDEs

A 2 variable LDE characterizes pair of 1 dimensional array access in a single non-nested loop. Refer to Figure 1 and LDE $2x - 3y = 5$. All solutions of the LDE can be derived using parametric solution {(-5 + 3m , -5 + 2m)}, where $-1 \leq m \leq 4$ is an integer[1]. If (x,y) is a solution of LDE and x precedes y in iteration order then x is called as source and y is called the sink. Solution (1,-1) obtained for m = 2, corresponds to $(source, sink) = (-1, 1)$. Similarly for m = 3, $(source, sink) = (1, 4)$. Since 1, source in (1,4) and sink in (-1,1), is an iteration common to both solutions, {-1,1,4} forms a C. Note that (8.5,4) and (-1,-2.33) are solutions, but are not integer

solutions, so they do not extend this C. Iteration 2 is neither a sink nor a source, hence 2 is an independent iteration.

Following observations can be made, a) an iteration which is a source as well as a sink extends C, b) an iteration which is either a source or a sink, but not both, does not extend C and c) an iteration which neither has a source nor a sink is an independent iteration.

3.1 Formation of Components

The objective of this section is to formulate the structure of a C and relationship between its constituent iterations. The solutions, to LDE in parametric form, are $\{(\alpha - mb), (\alpha + ma)\}$, where m is an integer and $\{\alpha, \alpha\}$ is a particular solution form Cs of length 2.

Longer Cs are formed if m is a multiple of a or b (m = qa or m= qb). As $(\alpha+qab)$ is a common iteration of $\{(\alpha - qbb), (\alpha + qab)\}$ and $\{(\alpha + qab), (\alpha - qaa)\}$ longer C = $\{(\alpha-qbb), (\alpha+qab), (\alpha-qaa)\}$ is formed.

The general structure of Cs of length $l + 1$ where m is not a multiple of a or b is: $\{(\alpha \pm ma^l), (\alpha \mp ma^{l-1}b), (\alpha \pm ma^{l-2}b^2), ..., (\alpha \pm (-1)^{l-2}.ma^2b^{l-2}), (\alpha \pm (-1)^{l-1}.mab^{l-1}), (\alpha \pm (-1)^l.mb^l)\}$

An iteration i of a component C is defined to be a seed of C if all the iterations of C can be generated using i. Seed is a representative iteration of a C which can generate it. For a single LDE single LOOP case, all iterations are seeds of corresponding Cs. If α is not an integer, C and seeds are computed using particular solution $\{\beta, \gamma\}$ of LDE as: $\{(\beta - mb), (\gamma + ma)\}$.

3.2 Bounds on $|P|$ and $|C|$

This sub-section shows computation of bounds on number ($|P|$) and length ($|C|$) of Cs for LDE in a given range R = [L,U]. The notation $|S|$ denotes the cardinality of set S.

Lemma 3.1. *Upper bound on number of Cs of length 2 is $|R|/|b|$.*

Proof. Every solution of LDE is a C of length 2. The maximum number of solutions in the range R is $|R|/|b|$. Hence proved. □

Number of Independent Iterations = $|R| - 2|R|/|b|$.

Lemma 3.2. *Upper bound on number of Cs of length $l = |R|/(|b|^{l-1})$.*

Proof. Proof by Induction: Upper bound on number of Cs of length 2 is $|R|/|b|$ using lemma 3.1.
Assumption: Upper bound on number of Cs of length k is $|R|/(|b|^{k-1})$.
To Prove That: Upper bound on number of Cs of length $k + 1$ is $R/(|b|^k)$.
An end element of a C of length k+1 has a form $(\alpha \pm qa^k) or (\alpha \pm qb^k)$.
if $q = ra$ where r is in integer the element $(\alpha \mp rb^{k+1})$ adds to the C. Similar addition takes place if q is a multiple of b.
The maximum number of elements having pattern $(\alpha \mp kb^k) \in$ [LB,UB] is $|R|/(|b|^k)$. Hence Proved. □

Let p_{max} be a positive integer such that $|R|/(|b|^{p_{max+1}}) < 1 \leq |R|/(|b|^{p_{max}})$ then the upper bound on length of Cs is $L_{max} = p_{max} + 1$.

The upper bound on Number of Cs is: $N_{max} = |R|/|b| - |R|/(b^2) + |R|/(|b|^3) - |R|/(b^4)...|R|/(|b|^{p_{max}})$
$= |R|\Big(|b|^{p_{max}} - (-1)^{p_{max}})/(|b|^{p_{max}}.(|b| + 1)\Big)$

This gives a closed-form expression for the bounds, a useful input for scheduling.
The length of longest sequential component is L_{max}. The Partition will have Cs of length up to L_{max}.
C of length l are built over 2 Cs of length l-1, 3 Cs of length l-2 ...etc.
This leads to a tight bound on number of Cs of length l.
Number of Cs of length l = $|R|/(|b|)^{l-1} - 2|R|/(|b|)^l - 3|R|/(|b|)^{l+1}... - (L_{max} - l)|R|/(|b|)^{L_{max}-1}$

3.3 Heuristic based IS splitting

Coefficients of LDE can either have same sign or different signs. If coefficients have same sign then large number of dependence edges cross iteration $\alpha = c/(a + b)$. This structure is of recurrence chains[7] but our heuristic uses a different observation. For example consider the following loop and its corresponding LDE x + 2y = 3.

```
for(i = -8; i <= 10; i++)
{ h[i] = .....;
  .... = h[3 - 2i]; }
```

The solution of LDE for the range R = [-8,10], in (sink,source) form is: { (-7,5), (-5,4), (-3,3), (-1,2), (**1,1**), (3,0), (5,-1), (7,-2), (9,-3) }. The

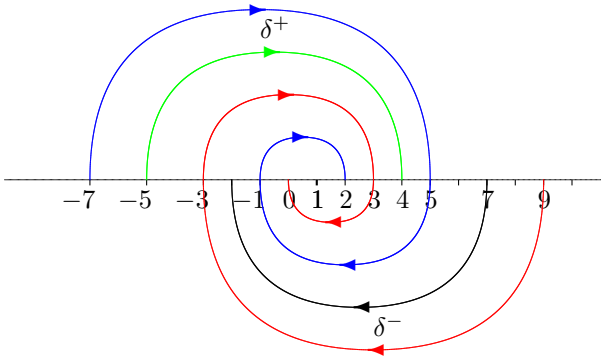

Figure 2: Cs of x+2y=3

dependence is as shown in Figure 2. The partition P = { {-7,5,-1,2}, {-5,4}, {-3,3,0,9}, {**1**}, {-2,7} }.

For such cases, if iteration $\alpha \in$ [LB, UB], distributing the Iteration range [LB,UB] into 2 ranges i.e. [LB,α] and [α+1,UB], largely improves the exploitable parallelism. In case of multiple 2 variable LDEs a heuristic based approach for efficient splitting is used. If αs of multiple 2 variable LDEs are in the range $[L_\alpha, U_\alpha]$, a suitable splitting point α_h will be $(L_\alpha + U_\alpha)/2$. The results of such a splitting are presented in section 8, Table 1.

4 PD for loop nest

We characterize a system of LDEs that covers a wide range of array subscripts found in loops. The LDE solved in section 3 is named as Type 1. The other 2 Types are introduced in this section. Presented below is an example of a loop nest that covers all 3 types of array access.

```
for( i = 0; i < 100 ; i++)
{ b[2i + 1] = ....
    .....= b[3i + 6];
for( j = 0; j < 100 ; j++)
{ a[2i + 1][2j + 4] = c[i + 2j];
c[2i + 3j + 6] = a[i][3j];   }   }
```

The example shows 2 loops with 2 loop indices i corresponding to outer loop and j corresponding to inner loop used for accessing arrays a,b and c. The different types of dependences are defined here.

Type 1(2 variable LDE): The dependence relation corresponding to single dimensional array access, having single loop index in its subscripts is represented by a 2 variable LDE. For example, access of array b is represented by LDE: $2x_1 - 3y_1 = 5$.

Type 2(multi variable LDE): The dependence relation corresponding to single dimensional array access, having multiple loop indices in its subscripts is represented by multi variable LDE. For example, access of array c is represented by 4 variable LDE: $x_1 - 2y_1 + 2x_2 - 3y_2 = 6$. In this LDE x_1, y_1 correspond to index i and x_2, y_2 correspond to index j.

Type 3(System of LDEs): Multiplicity in either dimension or number of array accesses is represented by a system of LDEs.
3a: The dependence relation corresponding to multidimensional array access in the nested loop is represented by one LDE per dimension, forming a system of LDEs. For example, access of array a is represented by system of 2 LDEs: $2x_1 - y_1 = -1$ and $2x_2 - 3y_2 = -4$.
3b: The dependence relation corresponding to access of multiple arrays in the nested loop is also represented by a system of LDEs. These arrays may belong to Type 1, Type 2 or Type 3a categories stated above. For example, accesses of arrays a & c are represented by LDEs: $x_1 - 2y_1 + 2x_2 - 3y_2 = 6$, $2x_1 - y_1 = -1$ and $2x_2 - 3y_2 = -4$.

Type 3 is the most general case. The complete solution requires merging solutions of all LDEs. Our approach extends the solution for Type 1 to solve Type 2. Our method progressively constructs solution of Type 3 LDE using those of Types 1 and 2.

4.1 Precise Dependence

Given an iteration space and a set of LDEs, we use the term Precise Dependence(PD) to denote the collection of all solutions of all LDEs in the iteration space. Iterations which do not appear in PD are independent iterations.
We construct and prove precise dependence theory for Type 1 LDEs and then extend it to other Types. Precise dependence information is obtained by solving LDEs and obtaining all the dependences in the given range of the loop. PD captures dependence exactly and hence all the parallelism that is present at iteration level granularity.
PD is represented as partition(P), i.e. set of Cs. A C in turn can be captured by its seed.
The method presented in the paper is termed as "Precise Dependence based Partitioning(PDP)" and can be used to generate P which may be used further for scheduling iterations. Scheduling is however, not in the scope of this paper.
Even though we present a general method for partitioning loops, we demonstrate that some special cases have interesting properties which can be ex-

ploited for efficiency and effectiveness.

4.2 PD for 2 variable LDE

The following 2 results, known for 2 variable LDE L and its solution S, provide the basis for the correctness of PDP for Type 1 LDEs.

1. All the dependent iterations in L are elements of S. (i.e if iterations i_1 and i_2 are dependent then $(i_1, i_2) \in S$ or $(i_2, i_1) \in S$.
2. All elements in S represent dependent iterations in L. i.e if $(i_1, i_2) \in S$ or $(i_2, i_1) \in S$ then iterations i_1 and i_2 are dependent.

5 Multi variable LDEs

Consider a general structure of a loop nest of Type 2 category as shown below.

$for(i_1 = lb_1; i_1 < ub_1; i_{1++})$
$for(i_2 = lb_2; i_2 < ub_2; i_{2++})$
.
.
$for(i_n = lb_n; i_n < ub_n; i_{n++})$
$\{S1 : g[a_{11}i_1 + a_{12}i_2 + .. + a_{1n}i_n] = ...;$
$S2 : .... = g[a_0 - a_{21}i_1 - a_{22}i_2 - .. - a_{2n}i_n];\}$

The LDE L representing dependencies in the loop nest is:

$$a_{11}x_1 + a_{21}y_1 + a_{12}x_2 + a_{22}y_2 ... a_{1n}x_n + a_{2n}y_n = a_0 \quad (1)$$

The LDE is constructed as follows:

- $I_x = \{x_1, x_2, .., x_n\}$ is source and $I_y = \{y_1, y_2, .., y_n\}$) is sink.
- Number of variables in LDE = 2 * number of distinct loop indices in the subscript.
- If a loop index is not part of an array subscript then its corresponding coefficient is zero.
- If an array is accessed exactly once in a loop nest then dependence and corresponding LDE does not exist.

A specific iteration in the loop nest is a vector of specific values of loop indices $I = \{v_1, v_2, .., v_n\}$.

5.1 PD for multi variable LDE

Precise Dependence in multi variable LDE is essentially a dependence between 2 iteration vectors. We solve a multi variable LDE by converting it to an equivalent set of 2 variable LDEs. The solutions of these sets of LDEs are combined to generate the solution of multi variable LDE.

Given $a_1x + a_2y = [L, U]$, note that this corresponds to a system of $U - L + 1$ Type 1 LDEs, $a_1x + a_2y = L$, ... $a_1x + a_2y = U$. There are few interesting results that allow us to construct the solutions of system of LDES by actually solving much lesser number of LDEs.

A: Translation The solutions of an LDE can be used to generate the solutions of many LDEs by translation. This is illustrated by the following.

Consider a pair of LDEs: $a_1x + a_2y = c$ and $a_1x + a_2y = c + a_1 + a_2$.
If (x', y') is a solution of $a_1x + a_2y = c$ then $(x'+1, y'+1)$ is a solution of $a_1x + a_2y = c + a_1 + a_2$.

Thus solution (x', y') can be translated to generate solutions $(x' + 1, y' + 1)$, $(x' + 2, y' + 2)$, $(x' + 3, y' + 3)$ etc for LDEs having RHS equal to $c + a_1 + a_2$, $c + 2a_1 + 2a_2$, $c + 3a_1 + 3a_2$ respectively.

This concept of translation is used to generate solutions of (5) from its subset. Lemma 5.1 shows correctness of this expanded range.

B:

Lemma 5.1. *Let S be solution of* $a_1x + a_2y = [L, U]$ *over the iteration range* $[l, u]$ *Let* S^x *be solution of* $a_1x + a_2y = [0, |a_1 + a_2|]$ *over the expanded iteration range* $[l - \frac{U}{|a_1+a_2|}, u - \frac{L}{|a_1+a_2|}]$
Then S can be generated by translating S^x.

Proof. Assume in general $L \leq 0$ and $U > |a_1 + a_2|$. S^x can be translated to generate solutions S^k for RHS values: $\forall k[k(|a_1 + a_2|), (k + 1)(|a_1 + a_2|) - 1]$, where $k \in \mathbb{Z}$ and $\frac{L}{|a_1+a_2|} \leq k \leq \frac{U}{|a_1+a_2|}$.
The resultant solutions will have iteration range $[l + k, u + k]$. Solutions S^k are required over iteration range $[l, u]$.
If $k > 0$ $[l, l + k - 1]$ is not included in S^k.
If $k < 0$ $[u + k + 1, u]$ is not included in S^k.
S^x computed over iteration range $[l - \frac{U}{|a_1+a_2|}, u - \frac{L}{|a_1+a_2|}]$ will include the final desired solutions. □

The solutions obtained for expanded range are pruned to required range. In Step 3 (6) gives the set of LDEs to be solved. Other solutions for (5) are obtained by translating solutions of (6).

C: GCD Applying GCD test[6] further reduces the set of LDEs to be actually solved as well as the set of LDEs to be solved by translation.

If $g = gcd(a_1, a_2)$ and m an integer such that $mg < a_1 + a_2 \leq (m+1)g$ then

$$a_1x + a_2y = [0, g, 2g, .., mg] \quad (2)$$

$$a_1x + a_2y = [L.. - 2g, -g, 0, g, 2g..U] \quad (3)$$

Partitions for (**??**) are computed using method given in section 3.1 .

The focus of research presented here is LDEs depicting non-uniform or variable distance data dependence. Based on coefficients of x and y we put forth two more possible categories for completeness.

1 If the product of coefficients is -1 it is a case of Constant or Uniform distance data dependence[6]. P contains 'c' Cs of length Range/c each if c$\neq$ 0. The structure of P is:
$P = \{\{l, l+c, l+2c, ...\}, \{l+1, l+1+c, l+1+2c, ...\}, \{l+2, l+2+c, l+2+2c, ...\}, ..., \{l+c-1, l+2c-1, l+3c-1, ...\}\}$
If c = 0, loop carried dependence does not exist and all iterations are independent.

2 If the product of coefficients is 0, then the entire range corresponding to the loop is totally dependent (on single array position). Such a P has a single C over the complete range:
$P = \{\{l, l+1, l+2, ..., u\}\}$

The actual steps of the method are:
For each loop index i_j, execute Steps 1 to 4.

Step 1. Rearrange the Type 2 LDE (1) by retaining terms corresponding to one loop index to left. For loop index i_j the modified LDE is:

$$a_{1j}x_j + a_{2j}y_j = c_j = a_0 - \sum_{k=0}^{k=n} (a_{1k}x_k + a_{2k}y_k), where\ k \neq j \quad (4)$$

Step 2. Apply Banerjee's Inequality[6] to compute the range $[L_j, U_j]$ for the RHS of (4) to get (5).

$$a_{1j}x_j + a_{2j}y_j = [L_j, U_j] \quad (5)$$

Step 3. Generate solutions of a subset of (5) to generate the solutions of (5) by translation and pruned to required range. (6) gives the set of LDEs to be solved. Other solutions for (5) are obtained by translating solutions of (6).

$$a_{1j}x_j + a_{2j}y_j = [0, |a_{1j} + a_{2j}| - 1] \quad (6)$$

Step 4. Applying GCD test[6] and reduce the set of LDEs to be actually solved (7) as well as the set of LDEs to be solved by translation (8). For m an integer such that $mg < a_{1j} + a_{2j} \leq (m+1)g$ then

$$a_{1j}x_j + a_{2j}y_j = [0, g, 2g, .., mg] \quad (7)$$

$$a_{1j}x_j + a_{2j}y_j = [L_j .. - 2g, -g, 0, g, 2g..U_j] \quad (8)$$

Compute Partitions for (8).

Step 5. The Ps of set of 2 variable LDEs obtained are merged to obtain iteration vectors of m variable LDE. The merging algorithm is:

1. Sort elements of Cs of 2 variable LDEs based on their distance from corresponding α's.
2. Merge Cs iteratively taking one from P of each LDE at a time as follows:
   (a) Take a consecutive pair of elements from each C at a time.
   (b) If ratio of absolute values of coefficients ≥ 1 add the iterations to corresponding iteration vectors in same order.
   (c) Else if ratio of absolute values of coefficients < 1 add the iterations to corresponding iteration vectors in reverse order.

Step 6. The resultant partition is computed as:
Initialize $P = \phi$
$P^r \uplus P^q$ is computed as follows
$\forall r \forall q \forall i \forall j$:
If$(C_i (\in P^r)\ \psi\ C_j (\in P^q))$ Add $C_i \cup C_j$ to P
Else Add C_i , C_j to P

The result of these 6 steps is the desired solution. Illustrations for steps 5 and 6 are given below.

The Ps of set of 2 variable LDEs thus obtained are merged to obtain iteration vectors of m variable LDE. This is illustrated using a 4 variable LDE (9):

$$a_{11}x_1 + a_{21}y_1 + a_{12}x_2 + a_{22}y_2 = a_0 \quad (9)$$

Let (p,q,r,s) be a solution to (9).
$\therefore$ (p,q) and (r,s) will be solutions to 2, 2 variable LDEs: $a_{11}x_1 + a_{21}y_1 = c_1$ and $a_{12}x_2 + a_{22}y_2 = c_2$ respectively, where $c_1 + c_2 = a_0$.

Possible dependences are (p,r)$\rightarrow$(q,s), (p,s)$\rightarrow$(q,r), (q,r)$\rightarrow$(p,s) and (q,s)$\rightarrow$(p,r). Only one of the above is precise dependence.

Consider the 2 variable LDE $a_{11}x_1 + a_{21}y_1 = c_1$ and its solution (p,q). Let $\alpha_1 = c_1/(a_{11} + a_{21})$. If $|a_{11}/a_{21}| \geq 1$ then $|p - \alpha_1| \leq |q - \alpha_1|$ and $|p - \alpha_1| \geq |q - \alpha_1|$ otherwise.

Similarly if in (9) ($|a_{11}/a_{21}| \geq 1$ and $|a_{12}/a_{22}| \geq 1$) OR ($|a_{11}/a_{21}| < 1$ and $|a_{12}/a_{22}| < 1$) the required

solution is (p,r)→(q,s) and (p,s)→(q,r) other wise.

Consider the illustration of (9) shown in step 5, multiple sets of values of $c_1 + c_2 = a_0$ exists and will result in multiple solutions and partitions. If q such partitions are obtained for q sets of values of c_js they are then merged by $\uplus$. i.e.
$P = P^1 \uplus P^2 \uplus .... \uplus P^q$
The Cs in resultant partition may overlap. The overlapping Cs are dependent hence need to be merged. $\uplus$ merges these overlapping Cs and generates a union of all Cs. We define two binary boolean functions ψ and @ to evaluate the overlap.

@ detects overlap between 2 Iteration vectors I^x and I^y having m positions and values, $v^x_m \in I^x$ and $v^y_m \in I^y$
I^x @ I^y = true, if $\exists m\ v^x_m = v^y_m$
I^x @ I^y = false otherwise

ψ detects overlap between 2 Cs.
$C_i \psi C_j$ = true,
if $\exists\ I^x \in C_i$ and $I^y \in C_j$ such that I^x @ I^y = true
$C_i \psi C_j$ = false otherwise

The Steps 1 to 4, which use results of Type 1 LDEs are shown to be correct. The Steps 5 ans 6 are constructive in nature are intutive and can be proved.

5.2 Illustrative example

This section shows working of examples based on the method shown in previous section.

E.g. Consider the following loop-nest:
$for(i = -3; i < 1; i++)$
$for(j = 0; j < 2; j++)$
$\{g[2i + 3j + 1] = ...;$
$..... = g[i + 4j + 4];\}$

The LDE to be solved is $2x_1 - y_1 + 3x_2 - 4y_2 = 3$ and steps are:

1. Modified LDE pair as per Step1:
   (a) $LDE_1 : 2x_1 - y_1 = c_1 = 3 - 3x_2 + 4y_2$
   (b) $LDE_2 : 3x_2 - 4y_2 = c_2 = 3 - 2x_1 + y_1$
2. The set of LDEs obtained as per Step 2 are:
   (a) $LDE_1 : 2x_1 - y_1 = [0, 7]$
   (b) $LDE_2 : 3x_2 - 4y_2 = [0, 9]$
3. The specific set of LDEs for which $c_1 + c_2 = 3$ are:
   (a) $LDE^0_1 : 2x_1 - y_1 = 0$, $\alpha_1 = 0$ &
   $LDE^3_2 : 3x_2 - 4y_2 = 3$, $\alpha_2 = -3$
   (b) $LDE^1_1 : 2x_1 - y_1 = 1$, $\alpha_1 = 1$ &
   $LDE^2_2 : 3x_2 - 4y_2 = 2$, $\alpha_2 = -2$
   (c) $LDE^2_1 : 2x_1 - y_1 = 2$, $\alpha_1 = 2$ &
   $LDE^1_2 : 3x_2 - 4y_2 = 1$, $\alpha_2 = -1$
   (d) $LDE^3_1 : 2x_1 - y_1 = 3$, $\alpha_1 = 3$ &
   $LDE^0_2 : 3x_2 - 4y_2 = 0$, $\alpha_2 = 0$
4. Minimal set of LDEs to be solved and corresponding solutions as per Step 4 are:
   (a) $2x_1 - y_1 = Cfinal_1 = [0]$ &
   $P_1 = \{\{-6,-3\},\{-5\},\{-4,-2,-1\},\{0,0\}\}$
   (b) $3x_2 - 4y_2 = Cfinal_2 = [0]$ &
   $P_2 = \{\{0,0\},\{1\},\{2\},\{3,4\}\}$
5. The partitions for sets of value pair for which LDEs have to be solved as per Step 5 are:
   (a) Solution of LDE: (-1,-2,1,0) & (0,0,1,0)
   $P^0_1 = \{\{-3\}, \{-1, -2\}, \{0, 0\}\}$ translated by 0
   $P^3_2 = \{\{0, 1\}\}$ translated by -3
   $Resultant : P^{0+3} =$
   $\{\{(-3, 0)\}, \{(-3, 1)\}, \{(-2, 0), (-1, 1)\},$
   $\{(-2, 1)\}, \{(-1, 0)\}, \{(0, 0), (0, 1)\}\}$
   (b) Solution of LDE: (0,-3,0,0)
   $P^3_1 = \{\{0, -3\}, \{-2\}, \{-1\}\}$ translated by +3
   $P^0_2 = \{\{0, 0\}, \{1\}\}$ translated by 0
   $Resultant : P^{3+0} =$
   $\{\{(-3, 0), (0, 0)\}, \{(-3, 1)\}, \{(-2, 0)\},$
   $\{(-2, 1)\}, \{(-1, 0)\}, \{(-1, 1)\}, \{(0, 1)\}\}$
6. The final partition is:
   $P = \{\{(-3, 0), (0, 0), (0, 1)\}, \{(-3, 1)\},$
   $\{(-2, 0), (-1, 1)\}, \{(-2, 1)\}, \{(-1, 0)\}\}$

In this section we have presented a method of solving multi variable LDEs by extending method to solve 2 variable LDE. Our method computes detailed solution using method to solve 2 variable LDE for non-trivially small range and expands it to the complete solution using translation, pairwise selection and a non trivial union by $\uplus$.

6 System of LDEs

Dependence information for a multi dimensional array access or multiple single dimensional arrays nested in multiple loops will result in a system of LDEs. Consider the loop

$for(i_1 = lb_1; i_1 < ub_1; i_{1++})$

$for(i_2 = lb_2; i_2 < ub_2; i_{2++})$
$\{S1 : g[2i_1 + 3i_2][3i_1 - 2i_2] = ..;$
$S2 : .. = g[i_1 + i_2 + 2][i_1 - i_2 - 1];$
$S3 : .. = g[i_1 - i_2 + 1][2i_1 + i_2];$

The following LDEs represent dependence existing in the loop

$$2x_1 - y_1 + 3x_2 - y_2 = 2 \quad (10)$$
$$3x_1 - y_1 - 2x_2 + y_2 = -1 \quad (11)$$
$$2x_1 - y_1 + 3x_2 + y_2 = 1 \quad (12)$$
$$3x_1 - 2y_1 - 2x_2 - y_2 = 0 \quad (13)$$

P_{10} and P_{11} are partitions of LDEs of different subscripts of same array g for reference pair in statements S_1 and S_2. Dependence exists if P_{10} and P_{11} are satisfied simultaneously. Hence the resultant partition is: $P^{1-2} = P_{10} \cap P_{11}$.
P_{12} and P_{13} are partitions of LDEs for reference pair in S_1 and S_3. Hence the resultant solution is: $P^{1-3} = P_{12} \cap P_{13}$.
The final partition will require merging of $P^{1-2} \uplus P^{1-3}$ as shown in Step 6 of 5.1.

Consider a general structure of a loop nest of Type 3 category as shown below.

$for(i_1 = lb_1; i_1 < ub_1; i_{1++})$
$for(i_2 = lb_2; i_2 < ub_2; i_{2++})$
:
$for(i_n = lb_n; i_n < ub_n; i_{n++})$
$\{S1 : g[a_{11}i_1 + a_{12}i_2 + .. + a_{1n}i_n] = ..;$
$S2 : .. = h[b_{11}i_1 + .. + b_{1k}i_k][b_{1j}i_j + .. + b_{1n}i_n - b_{20}];$
$S3 : h[b_{10} - b_{21}i_1 - .. - b_{2k}i_k][b_{2j}i_j - .. - b_{2n}i_n] = ..;$
$S4 : .. = g[a_0 - a_{21}i_1 - a_{22}i_2 - .. - a_{2n}i_n];\}$

(14) - (16) form a system of LDEs representing dependencies in the loop nest.

$$a_{11}x_1 + a_{21}y_1 + a_{12}x_2 + a_{22}y_2 ... a_{1n}x_n + a_{2n}y_n = a_0 \quad (14)$$
$$b_{11}x_1 + b_{21}y_1 + b_{12}x_2 + b_{22}y_2 .. b_{1k}x_k + b_{2k}y_k + 0x_{k+1} + 0y_{k+1} + 0x_{k+2} + 0y_{k+2} + .. + 0x_n + 0y_n = b_{10} \quad (15)$$
$$0x_1 + 0y_1 + 0x_2 + 0y_2 .. 0x_{j-1} + 0y_{j-1} + b_{1j}x_j + b_{2j}y_j .. + b_{1n}x_n + b_{2n}y_n = b_{20} \quad (16)$$

The complete solution is $P_{14} \uplus (P_{15} \cap P_{16})$

6.1 Method to Solve system of LDEs

For computing PD in case of system of LDEs, Ps of individual LDEs need to be computed and then merged. In a loop nest individual multi variable LDEs are solved using method shown in section 5.1. PDP Algorithm for system of LDEs is as follows:

1. For every k^{th} pair of n dependent array accesses:
   (a) For every i^{th} position of m subscripts.
      i. Construct LDE_i.
      ii. Generate partition P_i for LDE_i using method given in 5.1.
   (b) $P_k = \forall i \; P_1 \cap P_2 \cap .. P_i \cap .. P_m$
2. $P_{sys} = \forall k \; P_1 \uplus P_2 \uplus .. P_k \uplus .. P_n$

6.2 Illustrative example

This section shows working of examples based on the method shown in previous section.

E.g. 1 Consider the following loop-nest:
$for(i = -3; i < 1; i++)$
$for(j = 0; j < 2; j++)$
$\{g[2i + 3j + 1] = ..d[3i - 2j - 2]..;$
$d[2i + j + 3] = ..g[i + 4j + 4]..;\}$

The steps of solution are
Step 1: System LDEs
LDEg: $2x_1 - y_1 + 3x_2 - 4y_2 = 3$ &
LDEd: $3x_1 - 2y_1 - 2x_2 - y_2 = 5$
Step 2: Partitions of individual LDEs
$P_g = \{\{(-3, 0), (0, 0), (0, 1)\}, \{(-2, 0), (-1, 1)\}\}$
$P_d = \{\{(-3, 1), (0, 0)\}\}$
Step 3: Final Partition $P_{sys} = \{\{(-3, 1), (-3, 0), (0, 0), (0, 1)\}, \{(-2, 0), (-1, 1)\}\}$

7 Comparative Analysis

Contemporary papers that deal with VD and the published approaches have been studied and reviewed. The paper by Yu and D'Hollander[5] uses unimodular transforms for identifying parallelizable loops. Another paper by the same authors[7] uses recurrence chains for parallelization. Work done by Anna et al is based on slicing[9]. In PDP $|P|$ = number of parallel Cs and $|Ind|$ is number of independent iterations, larger value of $(|P| + |Ind|)$ shows better parallelization. $|C_{max}|$ is the size of longest C and its smaller value is desired. The results are compared with our approach and CLooG code generated by Pluto-0.11.3[11], which uses the polyhedral approach[8] are shown in Tables i, ii and iii. su and t_{max} stand for sequential unit and maximum parallel threads in the tables.
Example 1 [5]

```
for(i=-n;i<=n;i++)
  for(j=-n;j<=n;j++)
  { a(3i+1,2i+j-1) = ....
    .... = a(i+3,j+1)}
```

$\|IS\| = 441$	[5]	[11]	PDP
$\|P\|$	82	1	76
$\|Cmax\|$	6	41 su	3
$\|Dep\|$	222	t_{max}	178
$\|Ind\|$	219	$= 21$	263

Table i: Compared results for Example 1

Example 2 [7]

```
for(i=1;i<=n;i++)
  for(j=i1;j<=n;j++)
  { a(3i+1,2i+j-1) = ....
    .... = a(i+3,j+1)}
```

$\|IS\| = 144$	[7]	[11]	PDP
$\|P\|$	1	1	20
$\|Cmax\|$	avg 61 su	25 su	3
$\|Dep\|$	t_{max}	t_{max}	44
$\|Ind\|$	$= 36$	$= 22$	100

Table ii: Compared results for Example 2

Examples 3 to 5 [9]

```
Example 3: for(i=1;i<=n;i++)
             for(j=1;j<=m;j++)
             { a(i,j)=b(i,j)+c(i,j);
               c(i,j-1)=a(i,j+1);}

Example 4: for(i=1;i<=10;i++)
             for(j=1;j<=10;j++)
               a(j+4,j+1)=a(i+2*j+1,i+j+3)

Example 5: for(i=1;i<=n;i++)
             for(j=1;j<=n;j++)
               a(i,j)=a(2*i,1)-a(i,j+1);
```

IS=100	Eg4			Eg5	
	[9]	[11]	PDP	[9]	PDP
$\|P\|$	7	1	3	5	5
$\|C_{max}\|$	40	21 su	11	40	40
$\|Dep\|$	100	t_{max}	33	100	100
$\|Ind\|$	0	$= 18$	67	0	0

Table iii: Compared results for Examples 4 & 5

Example 3 has constant dependence results of all 3 methods match. The code for example 5 is kept sequential by Pluto-0.11.3[11].

8 Experimental Results

Our approach demonstrates that VD3 loops, in which dependence is represented by a few m-variable LDEs, not only possess reasonable parallelism, but it can also be exploited to partition the underlying iteration space.

	30LDEs		90LDEs	
Range	Avg	Heu.Avg	Avg	Heu.Avg
5	4.61	4.68	3.99	4.15
17	15.11	15.57	11.74	12.23
65	54.06	56.10	34.68	37.73
257	176.12	217.2	68.40	144.19
1025	525.85	880.85	114.74	660.92
4097	1557.49	3514.13	211.95	2651.60
16385	5488.09	14264.31	628.31	11192.73
65537	21179.73	57578.68	2199.60	46151.75

Table 1: Improved Average exploitable parallelism after loop splitting

However, it is not obvious, whether our observations about (a) the existence of parallelism scales to large number of LDEs with varying coefficients and iteration ranges and (b) whether PDP is useful in such situations.

Limited experiments were conducted using the parameters, i) loop bounds, ii) number of 2 variable LDEs, and iii) randomized coefficients for LDEs. The loop bounds were varied from $[-2, 2]$ to $[-2^{15}, +2^{15}]$, number of LDES from $[1, 90]$ and the results were averaged over 100 experiments. The results for 30 and 90 LDEs for 8 loop bounds are displayed in Table 1. Column labeled Avg, i.e. $avg(|P| + |ind|)$, is a measure of parallelism when PDP is used for partitioning. The Column labeled, Heu. Avg. shows $avg(|P|+|ind|)$ when the iteration space $[LB, UB]$ is sliced using the heuristic given in Section 3.3 into $[LB, \alpha_h]$ and $[\alpha_h + 1, UB]$. It can be seen that parallelism decreases with increase in number of LDEs, for 30 LDEs, parallelism drops from 4.61/5 to 21180/65537, providing lesser opportunity for PDP to exploit. However, PDP with heuristics, extracts significant parallelism even in these cases.

9 Conclusion

Partitioning of the iteration space of loops with non-uniform dependences is the focus of our research. The variability in the relation between a source and its sink makes the partitioning a difficult problem. We examined the parametric solution of a two variable LDE and identified the relation that exists between dependent iterations. The relation is used to create a partition which we call PD partition. We show that a component can be represented by a single iteration, called its seed. Our partition is named as PD partition because

we neither miss a dependence nor introduce any. The 2 variable LDE approach has been extended in stages to a multi variable LDE and then to a system of LDEs. We find a formulation of a multi variable LDE in terms of a set of 2 variable LDEs. Our formulation produces a number of 2 variable LDEs with similar structure, which is exploited to reduce the set of 2 variables LDEs that need to be solved explicitly. The PD partition for a multi variable LDE is constructed by appropriately merging the PD partitions of the equivalent set of 2 variable LDEs. PD partition for system of LDEs is constructed from the PD partition of the constituent LDEs along similar line.

The theory of PD developed by us establishes the feasibility of our approach. The complexity of solving an LDE, and hence PD partition, in the most general case is known. However, array access patterns and number of nested loops found in practice are small enough to be partitioned. The practicality of our approach is in partitioning VD loops commonly found in programs. Our experiments were aimed at evaluating scalability in presence of large number of LDEs. Our experiments show that parallelism exists and such loops can be beneficially partitioned. However, extent of parallelism reduces as the number of LDEs increase, which has been shown to improve using our heuristic. We have compared the quality of PD partition with the partitions produced by others[5, 7, 9, 11]. Our partition extracts more parallelism in case of VD loops.

PD partition can be generated at compile time. Given a multi core architecture, compile time generation of schedules for a PD partition would be the next step in automatic parallelization. Our handling of exploitable parallelism in VD expands the domain of applications that are parallelizable today.

References

[1] Utpal Banerjee. *Dependence Analysis for Supercomputing.* Kluwer Academic Publishers, 1988.

[2] C. D. Polychronopoulos. Compiler Optimizations for Enhancing Parallelism and Their Impact on Architecture Design. *IEEE Transaction on Computers, August 1988.*

[3] Ajay Sethi, Supratim Biswas and Amitabha Sanyal. Extensions to cycle shrinking. *International Journal of high Speed Computing*, 7(2):265–284, 1995.

[4] Utpal Banerjee. *Dependence Analysis.* Kluwer Academic Publishers, 1997.

[5] Yijun Yu and Eric H. D'Hollander Partitioning Loops with Variable Dependence Distances *International Conference on Parallel Processing*, pages 209-218. IEEE, August 2000.

[6] Randy Allen and Ken Kennedy. *Optimizing Compilers for Modern Architectures.* Morgan Kaufmann, 2002.

[7] Yijun Yu and Eric H. D'Hollander Non-uniform dependences partitioned by recurrence chains *International Conference on Parallel Processing*, August 2004.

[8] Uday Bondhugula, Albert Hartono, J. Ramanujam and P. Sadayappan A Practical Automatic Polyhedral Parallelizer and Locality Optimizer *Conference on Programming Language Design and Implementation*, June 2008.

[9] Anna Beletska, W Bieleki, A Cohen, M Palkowski and K Siedlecki Coarse-Grained Loop Parallelization: Iteration Space Slicing vs Affine Transformations *11th International Symposium on Parallel and Distributed Computing*, 2011.

[10] www.math.uh.edu/minru/5333-07/hj02cs01.pdf. *Summary: The Euclidean Algorithm and Linear Diophantine Equations.* 02.02.2014

[11] http://sourceforge.net/projects/plutocompiler/files/ *Pluto automatic parallelizer.* 23.03.2015