International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 4, Issue 2, ISSN No. 2455-2143, Pages 164-169
Published Online June 2019 in IJEAST (http://www.ijeast.com)

A REVIEW ON DEPENDENCE ANALYSIS AND PARALLELIZATION TECHNIQUES

Vijaya Balpande
Computer Science and Engg.
Priyadarshini J.L. College of Engineering

Pradnya Borkar
Computer Science and Engg.
Priyadarshini J.L. College of Engineering

Abstract—With the new era of multicore processors it is required to improve the execution speed of the program. Consequently, developing parallel programs is the primary concern of the multicore era. The main challenge is to convert the sequential algorithm to parallel. Conversion of the sequential algorithm to parallel program without changing the output requires analyzing the set of constraints called dependency. Dependency specifies the basis for powerful transformation systems that enhance the implicit parallelism present within the program. The applications that are implemented with data structures like trees, queues and graphs is the most difficult problem for parallelization as these data structure deals with pointer. Graph Isomorphism is a technique of matching the structure of one graph with another and if only subpart of a graph is matched it is known as subgraph isomorphism. Graph and subgraph Isomorphism is one of the most highly-studied problems in the various field of computer science. In this paper we have studied the data dependency, control dependency, different parallelization techniques and the graph isomorphism problem.

***Keywords*—Data dependency, Control dependency, Graph Isomorphism, Subgraph Isomorphism, Parallelization**

I. INTRODUCTION

With the evolution of parallel computers, the optimization process becomes too complex. The principal optimization becomes uncovering the parallelism in a sequential algorithm and tailoring the parallelism to the target machine. The basis of this approach is a dependence analysis. Dependence represents the set of constraints such as data dependence and control dependence. Data dependence constraint must ensure that the data is produced and consumed in correct order. In control dependence the order of execution of statements must be preserved depending on the condition specified in the program. For transformation, dependency analysis is a tool determining whether it is safe to transform the program that will preserve the same output of the original program. As specified in Allen et.al(2002) , a program transformation that changes the order of execution of statement without adding or deleting any statement in a program is known as reordering transformation. Reordering transformation thus preserve the relative execution order of source and sink of that dependence. For the two statements S and T, if the instance of statement T(i) depends

on instance S(i) then S(i) is called source and T(i) is called sink. For loop transformation, the loop iterations are needed to be standardized and the process of standardizing the loop iteration is loop normalization. In loop normalization, the index space is transform to iteration space to have unit stride. By loop normalization the non-uniform index space is transformed to uniform iteration space. The relation between the source and sink of dependence in the iteration space can be characterized by distance and direction vectors .For loop reordering transformation, direction vector can be used to specify the relationship between the index vector of the source and sink of dependence. The direction vector specifies in which direction the loop iteration are moving and distance vector specifies that the index variable value is increasing uniformly. For checking whether the parallelization of sequential program is possible or not dependence testing is required. For determining whether two references to the same variable in a given set of loop access the same memory location, dependence testing is required. Techniques for performing dependence testing are GCD test and Banerjee Inequality test as given in Allen et.al(2002). Dependence testing is carried out by the transformation techniques like loop normalization, constant propagation, and induction-variable. Loop normalization is performed to have uniform iteration space. For vectorization the loop interchange and wavefront method for parallelization are described in the papers of Lamport L et al.(1974, 1976,1981). Loop skewing which is the practical way of implementing the Lamport Wavefront method was introduced by Wolfe M.J (1986).The performance of parallelized code not only depend on parallelism found in the code but also on code being packed as fine grain and coarse grain granularity. Loop distribution is performed by node splitting which breaks the data dependency cycle. Loop Fusion is

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 4, Issue 2, ISSN No. 2455-2143, Pages 164-169
Published Online June 2019 in IJEAST (http://www.ijeast.com)

performed by merging two loops into a single loop. To apply loop fusion both loops must have same structure, loop depth, loop bounds and iteration direction. For optimization of array, Loop Methods combining loop distribution and Loop Fusion were discussed in Allen et. al.(1987).

An application like chemical compounds, geographic maps, computer networks, software systems architectures, social networks requires the information to be represented in the form of graphs. For example, for finding the relationship of a person in a social network, requires to process huge database which can be represented in the form of graph. Comparison between graphs can be formulated as a graph or subgraph isomorphism

Parallelizing the pointer based data structure requires analyzing the dependence structure in the sequential algorithm. Dependency analysis includes data dependence, control dependence, flow dependence. For exploiting parallelism of sequential algorithm the loop structure requires rigorous analysis. Dependence in loop structure influences the parallelization of sequential algorithm.

For parallelizing the numerical and scientific applications DOALL and DOACROSS techniques Allen et. al.(2002) are used for loop parallelization and performs well for on very regular and analyzable structure that has predictable array accesses but not suitable for unpredictable data access pattern.

In this paper section II describes the dependence analysis and its properties, section III describes the existing techniques of parallelization, section IV describes graph isomorphism problem and section V includes the conclusion.

II. DEPENDENCE AND ITS PROPERTIES

A. Data Dependence:

Data dependence arises due to use of same data in more than one statement and accesses the same memory location and at least one of the statement stores data into it.
For following program code 1:

S_1: r= 3.0
S_2: pi =
3.14
S_3: peri = 2 * pi * r

No execution constraint exist between statement $S_1$and S_2 as execution order S_1,S_2 ,S_3 and S_2 ,S_1 ,S_3 will produce the same result.

B. Control Dependence:

A dependence that arises due to control flow is called control flow dependence.
For following program code 2:

S_1: if (y! = 0) goto
S_3 S_2: x = x / y
S_3: continue

Executing S_2 before S_1 could cause a divide-by-zero exception. Therefore S_2 cannot be executed before S_1 as S_2 is conditionally dependent on S_1.

C. True Dependence or Flow Dependence:

For following program code 3:

S_1: y = a +
b S_2: z = x
+ y

The value y computed by S_1 is used by S_2. Dependence of S_2 on S_1 is called true dependence or flow dependence represented

by S_1 δ S_2. In S_1, y is used as an output variable and in S_2 , y is used as an input variable. Flow dependence is same as read after write (RAW) hazard.

D. Antidependence:

For following program code 4:

S_1: p = y +
b S_2: y = x
+ 3

The value y computed by S_2 is read by S_1.This prevents the interchange of S_1 to S_2 .This type of dependence is called antidependence represented by S_1 δ^{-1} S_2 and is equivalent to write after read (WAR) hazard.

E. Output Dependence:

For following program code 5:

S_1: y = a +
b S_2: y = x
+ z

Both the statements S_1 and S_2 uses y as an output variable and stores the value in the same variable. Statements S_1 and S_2 thus write into the same location. .This type of dependence is called output dependence represented by S_1 δ^o S_2 and is equivalent to write after write (WAW) hazard.

F. Dependence in Loops:

For following program code 6:

: DO I = 1, n
S_1: A (I+1) = A (I)
+B (I) ENDDO

In statement S_1, instance A(3) uses the value of A(computed in the previous iteration any loop iteration depends on the instance of itself executed in previous iteration. The iteration vector is denoted as

$$i = \{i_1, i_2 \ldots i_n\} \quad (1)$$

where i_k, $1 \leq k \leq n$, represents the iteration number for the loop at nesting level k.
For the following program code7:

DO I = 1, 2

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 4, Issue 2, ISSN No. 2455-2143, Pages 164-169
Published Online June 2019 in IJEAST (http://www.ijeast.com)

```
DO J = 1, 2
        S1
ENDDO
```

The set of all possible iteration vectors for a statement S_1 is an iteration space. The iteration space of S_1 is {(1,1),(1,2),(2,1) ,(2,2)}.

G. Reordering Transformation

For following program code 7:

```
S1:  r =
3.0 S2: pi
= 3.14
S3: peri = 2 * pi * r
```

Reordering the statement S_1, S_2, S_3 as S_2, S_1, S_3 will produce the same result. A transformation is valid if it preserves all dependences in the program. A reordering tranformation changes the order of execution of the statement without adding or deleting any executions of any statements and preserves the source and sink of transformation in Allen et. al.(2002).

H. Distance and Direction Vectors

For following program code 8:

```
L1:     do I1 = 10, 100, 3
L2:         do I2 = 50, 5, -2
S:              X (2I1 - 1, I1 + I2) = Y (I1 + I2)
T:              Z (I1 + I2) = X (3I2 + 1, 2I1 + 2)
            en
        ddo
        enddo
```

By formulating the equation for statement S and T we

get, $I_1 = i_1$, $I_2 = i_2$ S: X $(2i_1 - 1, i_1 + i_2)$

$I_1 = j_1$, $I_2 = j_2$ T: X $(3j_2 + 1, 2j_1 + 2)$

$$2i_1 - 1 = 3j_2 + 1 \quad (2)$$

$$i_1 + i_2 = 2j_1 + 2 \quad (3)$$

By rearranging the above equation 2 and 3 we get,

$$2i_l - 3j_2 = 2 \quad (4)$$

$$i_l - 2j_l + i_2 = 2 \quad (5)$$

Constraints on i_1, j_1, i_2, j_2 will be

$I_1 = \{10, 13 \ldots 100\}$

$I_2 = \{50, 48 \ldots 6\}$

In above program segment the index space of I_1 and I_2 have an arbitrary stride so to standardize the index space to have an unit stride so that the index variable to be increase in sequential order as 0,1,2, … loop normalization is carried out. Loop normalization is performed by introducing a new variable called iteration variable and new iteration space having unit stride is found. Dependence between statement instances, S(i) an instance of statement S is determined by an index point i, and T(j) the instance of statement T is determined by an index point j and the distance from S(i) to T(j) is written as distance between the source S(i) and sink T(j) of dependence in the iteration space of the loop nest containing the statement involved in the dependence. For loop normalization the distance and direction vector are needed to be found.

Dependence of T on S for the set of all pairs (S (i), T (j)) that satisfies the condition i < j such that iteration H(j) dependence on iteration H(i) and the distance vector is d, direction vector σ and dependence level *l* is calculated as stated in equation 7, 8 and 9.For i < j ,the distance vector d(i ,j) is defined as in Allen et. al.(2002).

$$d(i, j) = j - i \quad (7)$$

Since i ≤ j, distance vector must always be lexicographically non-negative. No legal dependence can have negative distance because this would indicate that the source of the dependence was executed before sink. This means the source and sink are reversed and they are antidependent.

The direction vector σ(i, j) is defined as

$$\sigma(i, j) = sign(d) \quad (8)$$

It is also specifies as (<, =,>) depending on the relative values of iteration vectors i and j .The arrow points to the loop iteration that occurs first pair of iteration vector s for the source and sink of the dependence.

Direction vectors can be used as a basis for understanding loop reordering transformations because they summarize the relationship between the index vectors at the source and sink.

The dependence level l is specified as

$$l = lev(d) \quad (9)$$

Dependence level are determined based on the value of i and j

.If i=1 and j=1 level is 1,if i=0 and j=1 then level is 2 and if both i = j =0 then level is 3.At level 3 loops are totally independent. If S_2 depends on S_1 at level l for $(1 \leq l \leq m)$, we say that dependence of S_2 on S_1 is carried by Li. Computation of distance vector, direction vector and dependence level for the program code 8 is specified.

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 4, Issue 2, ISSN No. 2455-2143, Pages 164-169
Published Online June 2019 in IJEAST (http://www.ijeast.com)

For following program code 9:

```
L1: DO I = 0, 4, 1
        DO J = 0, 4, 1
S1;         A (I + 1, J) = B (I, J) + C
(I, J) S2;  B (I, J+1) = A (I, J+1) + 1
S3:         D (I, J) = B (I,
        J+1) -2 ENDDO
    ENDDO
```

For array A dependence between S_1 and S_2can be specified as
,at I=2,J=2 the instance $S_1(2,2)$ writes A(3,2) and at I=3,J=1 , the instance of $S_2(3,1)$ reads A(3,2).Instance $S_1(2, 2)$ write is executed before the instance $S_2(3,1)$ read. The direction vector

$$S(i) - T(j) = (\hat{j} - \hat{i}) \quad (6)$$

level *l*=1. It indicates that S_2 is flow dependent on S_1 that is S_1
δ S_2.

where

i and j are the new iteration points corresponding to

For array B dependence between S_1 and S_2can be specified as, at I = 2, J = 2 the instance $S_1(2,2)$ reads B(2,2) and at I = 2, J = 1, index point i and j. Dependences can be characterized by the $S_2(2,1)$ writes B(2,2).the instance S_2 (2, 1) write is executed before $S_1(2,2)$ read. The direction vector d (2, 1) = (0, 1), distance vector σ = (1, 1) and dependence level *l* =2. It indicates that S_1 is flow dependent on S_2 that is $S_2\delta$ S_1.

I. Loop Interchange:

For improving the performance of program, loop interchange is the most useful transformation and helps in exploiting the parallelism in the loop.

L. Loop Fusion:

Loop fusion for following program code 12:

```
L1:     DO I = 1, N
          A (I) = B (I)
        + 1 ENDDO
L2:     DO I = 1, N
          C (I) = A (I) + C(I-1)
```

For following program code 10: DO I = 1, N

```
        DO J = I, M
S1;       B (I, J+1) = B (I, J) + 1
L3:
ENDDO
DO I = 1, N
  D (I) = A (I) + 1
ENDDO
        END
    DO
    ENDDO
```

True dependence is carried by the innermost loop itself. If loops are interchanged the dependence is carried by outer loop and inner loop remains dependence free and we achieve fine- grained parallelism. Rewriting the above program code

```
        DO J = 1,
          M DO I
          = I, N
S1;         B (I, J+1) = B (I, J)
          + 1 ENDDO
        ENDDO
```

For the coarse-grained parallelization, a parallel loop is moved to the outer most position to increase granularity and decrease synchronization overhead in Allen et. al.(2002). In loop interchange dependency between the statements changes. For the loop interchange the dependence vector must remain positive.

J. Node splitting:

d(2,2)=(1 ,1),distance vector σ(2,2)=(1 ,1) and dependence.

Node splitting is the loop transformation technique which breaks the data dependency cycle and helps to parallelize the loop.
For following program code 11:

```
        DO I = 1, N
S1;       A (I) = B (I+1) + B(I)
S2;       B (I+1) = C (I)
        + 5 ENDDO
```

in the above loop there are two different references to B in the statement S_1 and there exist anti dependence S_1 δ^{-1} S_2 and recurrence. This is removed by the technique called node splitting .Node splitting creates a copy of node from which an anti dependence emanates, if there are no dependence coming in the node and the recurrence is broken and the code can be rewritten as

```
        DO I = 1, N
          B (I) = B (I+1)
S1;       A (I) = B' (I) + B (I)
S2;       B (I+1) = C (I)
        + 5 ENDDO
```

K. Loop Skewing :

Loop skewing is a transformation that reshapes an iteration space and expresses the existing parallelism with conventional parallel loops.

Loops L_1 and L_3 carry no dependence so they can be merged together to increase the granularity and the code can be rewritten as

```
L1:     PARALLEL DO I = 1, N
          A (I) = B (I) + 1
L3:       D (I) = A (I)
        + 1 ENDDO
L2:     DO I = 1, N
          C (I) = A (I) + C
        (I-1) ENDDO
```

Loop transformation achieved by merging two loops together is known as loop fusion.

M. Unimodular transformation:

Loop interchange, loop skewing and loop reversal are examples of general set of transformation known as unimodular transformation. Unimodular transformation is implemented for only perfectly nested loops. It supports goal directed parallelization strategies.

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 4, Issue 2, ISSN No. 2455-2143, Pages 164-169
Published Online June 2019 in IJEAST (http://www.ijeast.com)

III. PARALLEIZATION TECHNIQUES

For extraction of parallelism for running it on multicore processor to improve the performance of algorithm the different parallelization techniques are tiling technique given in Tan et.al. (2007) and Wolf M.E.et.al. (1991), speculative parallelization explained in Raman Arun (2012), decoupled software pipelining Cintra et.al R(2005) and speculative decoupled software pipelining Cintra et.al R(2003).Tiling technique is used to exploit the parallelism in loop by analyzing the dependencies. For implementing the tiling loop skewing, loop transformation techniques are used. Tiling increases the granularity of computation and decreases the amount of communication incurred between the processors. It improves the data locality and data reusability which result in better utilization of cache locality. As the data remain in cache for the entire iteration, moving in and out of data required for computation is reduced thereby improving the overall performance. It also improves the register reuse.

Speculative Parallelization is implemented by analyzing the values of data dependencies between the different tasks and parallelizing the sequential code.

Decoupled software pipelining exploits the fine grained level parallelism as compared to DOACROSS parallelism specified in Allen et. al.(2002). In DOACROSS parallelism loops consist of dependencies among the iteration of the loop. DOACROSS parallelism is

characterized by the parallel execution of parts of each loop iteration across multiple cores. Dependences are handled by forwarding values from core to core by some way, often through memory with synchronization. In DSWP, there are no restrictions like control flow should be simple, to operate only on arrays and should have regular memory access pattern that can be seen in DOACROSS.DSWP partition the loop code and particular piece of code across all iteration is executed on each core for which the core is responsible. It results into long communication latencies between threads.

The problem of dependence recurrence present in DSWP can be handled by speculative decoupled software pipelining
Cintra et.al R(2003) .Speculative DSWP combines the speculation and pipeline parallelism and improves the significant speedup in the presence of long inter-core communication latency.

IV. GRAPH ISOMORPHISM

Graph Isomorphism is a technique of matching the structure of one graph with another and if only subpart of a graph is matched it is known as subgraph isomorphism. Graph and subgraph Isomorphism is one of the most highly-studied problems in the various field of computer science. Graph is used for representing information in computer networks, social networking, data mining, chemical compounds, etc. For enhancing the performance of sequential algorithm and exploiting the resources of multicore conversion of sequential program to parallel program is required. Subgraph Isomorphism is found to be solved in polynomial time but graph isomorphism is found not to be NP-complete. Despite of much effort no polynomial-time algorithm for graph isomorphism has been found. Several subproblems of graph isomorphism are known to have polynomial algorithms.

The graph isomorphism is expressed as in Deo Narsingh et al.(1995) : Given two graphs $G=(V_1,E_1)$ and $H=(V_2,E_2)$,if there exist one to one mapping function f from v_1 to v_2 such that (i , j) $\in E_1$, if and only if (f(i) ,f(j)) $\in E_2$. The function f is called an isomorphism from G to H. If the two graphs isomorphic to each other, it is denoted by $G \cong H$.

Exact Matching and Inexact Matching are the two ways for matching the graphs. Exact graph matching is characterized by mapping between the two nodes of two graphs if there is an edge between the two nodes in the first graph, they are mapped to two nodes in the second graph that are linked by an edge by preserving the edge. The different form of exact matching is graph isomorphism, subgraph isomorphism, monomorphism or Automorphism and Maximum common subgraph (MCS).For exact matching the different techniques exist like tree search based algorithms and canonical labeling. Ullmann Algorithm in Ullmann J. R et. al.(1997) , Messmer B.T. and Bunke H.(1995) , D.G.Corneil and Gotelib(1970) and Nauty Algorithm in McKay Brendan D et. al(1981,2004) and in are the algorithm based on tree search method and canonical labeling. In McKay Brendan D et. al(1981,2004) algorithm based on canonical labeling is described. Canonical labeling is practically available algorithm on site of the author specified in McKay Brendan D(2004) .

In Qiu et. al(2010) , for testing the runtime input graph with the model database graph vertex invariant and decision tree concept is implemented. The vertex invariants are used to partition the matrix of the graph before graph isomorphism detection. The vertex invariant property of graph, the size of decision tree is reduced as compared to Messmer et.al(1995). The technique is similar to breadth pruning technique which reduces the size of decision tree remarkably still the time complexity is almost equivalent.

Decision tree is the most widely used method for inductive conclusion and simple method for knowledge representation.

Graph isomorphism problem can be solved by decision tree method using the vertex invariant. Parallelization of sequential algorithm of graph isomorphism can be done as it has wide variety of application.

V. CONCLUSION

With the evolution of parallel computers, the optimization process becomes too complex. The principal optimization becomes uncovering the parallelism in a sequential algorithm and tailoring the parallelism to the target machine. Mostly today's algorithms are sequential, which basically perform operations in a sequential fashion. As the speed at which multicore processors operate has been improving at an exponential rate it is necessary to design an algorithm that specifies multiple operations. In order to solve a problem efficiently on a parallel machine, dependence analysis plays an important role in parallelizing the sequential code. Dependence analysis helps in implementing parallelization techniques.

VI. REFERENCES

[1] Allen Randy and Kennedy Ken(2002): Optimizing

Compilers for Modern Architectures- A Dependence – Based Approach, Morgan Kaufmann Publisher Inc,.

[2] Lamport L(1974).: The parallel Execution of DO Loops. Coomunication of the ACM ,17(2):83-93.

[3] Lamport L(August 1976,revised October 1981).: The Co ordinate Method for the practical execution of iterative DO Loops. Technical Report CA-7608-0221,SRI,Menlo Park,CA,.

[4] Wolfe M.J(August 1986).: Loop Skewing: The Wavefront Method revisited.International Journal of Parallel Programming 15(4): 279- 293,.

[5] Allen R.,Callahan D and .Kenned K(January 1987):.Automatic decomposition of scientific programs for parallel execution .In Conference Record of the Fourteenth ACM Symposium on Principles of Programming Languages,.

[6] Tan Guangming, Sun Ninghui , Gao Guang R(,2007). :A Parallel Dynamic Programming Algorithm on a Multicore Architecture. ACM SPAA'07, June 9-11.

[7] Wolf M.E., Lam M.S(1991).: A data locality optimizing algorithm. Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation, Toronto,Ontario, (1991); SIGPLAN Notices 26(6), June 26-28 pp. 30–44 .

[8] Raman Arun(2012): A System for Flexible Parallel Execution. Ph.D. Thesis, Department of Electrical Engineering, Princeton University.

[9] Cintra Marcelo, Llanos Diego R(2005).: Design Space Exploration of a Software Speculative Parallelization Scheme. IEEE Transactions On Parallel And Distributed Systems, Vol. 16, No. 6,

[10] Cintra Marcelo, Llanos Diego R.(2003): Toward Efficient and Robust Software Speculative Parallelization on Multiprocessors. ACM PPoPP'03, San Diego, California, USA June 11–13,.

[11] Deo Narsingh(1995): Graph Theory with Applications to Engineering and Computer Science ,Prentice Hall,Inc.

[12] Ullmann J. R(1997).: An algorithm for subgraph isomorphism. Journal of the ACM, 23(1):31–42.

[13] Messmer B.T. and Bunke H.(1995): Subgraph Isomorphism in Polynomial Time. University of Bern, Institute of Computer Science and Applied Mathematics, Bern, Switzerland Technical Report IAM 1995-003.

[14] Corneil D. G. and Gotlieb C. C(1970).: An efficient algorithm for graph isomorphism. Journal of the ACM, 17(1):51–64.

[15] McKay Brendan D(1981).: Practical graph isomorphism. Congressus Numerantium, 30:45–87.

[16] McKay Brendan D(2004,). . The nauty page. Computer Science Department, Australian National University, http://cs.anu.edu.au/bdm/nauty/.

[17] Qiu Ming , Hu Haibin, Jiang Qingshan and Hu Hailong(2010) : A New Approach of Graph Isomorphism Detection based on Decision Tree

IEEE, Second International workshop on Education Technology and Computer Science.