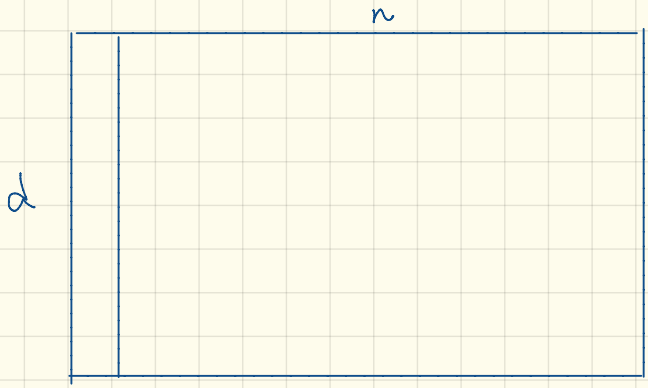
1. loop interchange, 2. permutation, 3. reversal, 4. hyperplane (skewing).
5 tiling and 6. concurrentization. can be realized as matrix multiplication by a suitable matrix.

- The order of applying a set of transformations has a great impact on the performance of the loop.
- Many loop transformations are linear transformations applied to the iteration space and the dependence matrix.

a dependence matrix → The distance vectors are the columns of the dependence matrix.



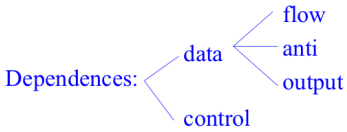
dependence matrix ($d \times n$) for d-nested loop

Goals

- determine what operations can be done in parallel
- determine whether the order of execution of operations can be altered

What should be the granularity of operations?

- depends on application
  - scheduling for pipelined machines: granularity = m/c instructions
  - loop restructuring: granularity = loop iterations

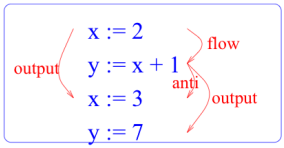


Flow dependence: S1 -> S2
(i) S1 executes before S2 in program order
(ii) S1 writes into a location that is read by S2

Anti-dependence: S1 -> S2
(i) S1 executes before S2
(ii) S1 reads from a location that is overwritten later by S2

Output dependence: S1 -> S2
(i) S1 executes before S2
(ii) S1 and S2 write to the same location

Input dependence: S1 -> S2
(i) S1 executes before S2
(ii) S1 and S2 both read from the same location



This theory provides the foundation for solving an open question in compilation for parallel machines: which loop transformations, and in what order, should be applied to achieve a particular goal, such as maximizing parallelism or data locality. This paper presents an efficient loop transformation algorithm based on this theory to maximize the degree of parallelism in a loop nest.

efficient use of the memory hierarchy. Existing vectorizing and parallelizing compilers focused on the application of *individual* transformations on pairs of loops: when it is legal to apply a transformation, and if the transformation directly contributes to a particular goal. However, the more challenging problems of

Including loop skewing in a "generate and test" framework is thus problematic, because a wavefront can travel in an infinite number of different directions.

This approach has been adopted particularly in the context of automatic systolic algorithm generation

Loop Nest Representation.

In the narrower domain of loops whose dependences can be represented as distance vectors.

directly in maximizing some objective function. Loop nests whose dependences can be represented as distance vectors have the property that an n-deep loop nest has at least $n-1$ degrees of parallelism [13], and can exploit data locality in all possible loop dimensions [24]. Distance vectors cannot represent the dependences of general loop nests, where two or more loops must execute sequentially. A commonly used notation for representing general dependences is *direction vectors* [2], [26].

a loop nest of depth n is represented as a finite convex polyhedron in the iteration space $\mathbb{Z}^n$ bounded by the loop bounds. Each iteration in the loop corresponds to a node in the polyhedron, and is identified by its index vector $\vec{p} = (p_1, p_2, \ldots, p_n)$; p_i is the value of the ith loop index in the nest, counting from the outermost to the innermost loop.

↓

The execution order of the iterations is constrained by their data dependences.

↓

Scheduling constraints for many numerical codes are regular and can be succinctly represented by dependence vectors.

what is dependence vectors ?

Abstract

Program parallelization is usually based on dependence graph analysis. The dependence graph is built on program statements and conveys ordering constraints on statements and statement iterations. These constraints must be summarized since statement iterations are virtually unbounded. This is usually done by using *Dependence Direction Vectors* (DDV). We introduce here a new concept called *Dependence Cone* (DC) that provides a more accurate dependence summary than DDV, and show that parallelization techniques based on DDV can be adapted to DC. DC's computation is based on linear systems and can exploit intra- and inter-procedural information on variable values and CALL effects. Furthermore DC's accuracy increases the number of possible reorderings with transformations like the hyperplane method and supernode partitioning and global parallelization.

The amount of information in :
- dependence vectors
- dependence direction vectors ?

Automatic program parallelization is not an easy task. Supercomputer architectures provide many features like vector registers or cache memory that cannot be exploited only by detecting parallel DO-loops in a program. Statements and statement iterations must be transformed and/or reordered without changing the program semantics. Reordering transformations must be compatible with Bernstein's conditions[4] and keep statement iterations which write and read the same memory location in the same temporal order.

4. A. J. Bernstein, "Analysis of Programs for Parallel Processing," *IEEE Transactions on Electronic Computers* **15**(5), pp.757-763 (Oct. 1966).

1. Basics of Parallelization. Assumptions and Notations.

Program parallelization and restructuring must preserve the initial program semantics. The order of read and write accesses to each memory location is kept while accesses to distinct memory locations can be exchanged. Thus the value history of each memory location remains the same and final states are equal at the bit level.

Dependence based parallelization enforces the order between any read and any write and between writes. This is a sufficient condition for histories to be preserved.

一个statement 会在不同的iteration 执行

空间

内存地址

顺序 ← iteration / statement / references < read ← 计算产生的副作用?

时间

write

vector

计算

references 刻划地是statement 的特性.

一个statement 会消耗掉一个由另一个statement 产生的值.

pure.

Each dependence vector defines a set of edges on pairs of nodes in the iteration space. Iteration $\vec{p_1}$ must execute before

(a)

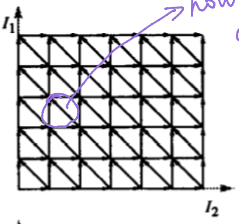
```
for I1 := 0 to 5 do
  for I2 := 0 to 6 do
    a[I2 + 1] := 1/3 * (a[I2] + a[I2 + 1] + a[I2 + 2]);
```

$D = \{(0,1),(1,0),(1,-1)\}$.

there are 6×7 points in the iteration space.

dependence vector

how these 2 edges come?



(b)

```
for I'1 := 0 to 5 do
  for I'2 := I'1 to 6+I'1 do
    a[I'2 − I'1 + 1] := 1/3 * (a[I'2 − I'1] +
      a[I'2 − I'1 + 1] + a[I'2 − I'1 + 2]);
```

$$T = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$D' = TD = \{(0,1),(1,1),(1,0)\}$

dependence vector in the transformed space.

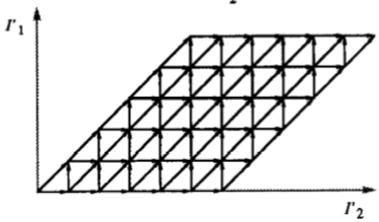
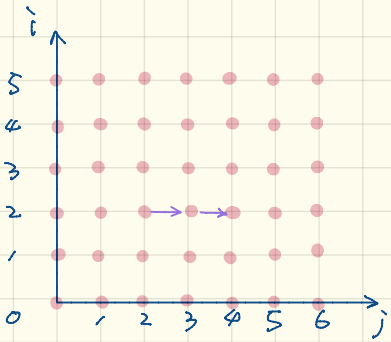


Fig. 1. Iteration space and dependences of (a) a source loop nest, and the (b) skewed loop nest.



Unimodular Transformations, perform a linear transformation on the iteration space.

$$T\vec{p}_2 - T\vec{p}_1 = T(\vec{p}_2 - \vec{p}_1)$$

Three important properties,

1. square: map n-d iteration space to n-d iteration space.
2. maps integer vectors to integer vectors
3. the absolute value of its determinant is 1. → what role does this property play?

✔ interchange, reversal, and skewing are useful in parallelization or improving the efficiency of the memory hierarchy.

✔ These transformations can be modeled as elementary matrix transformations.

1. Interchange: maps iteration (m, n, i, j, k) to (k, j, i, n, m)

In matrix notation,

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} m \\ n \\ i \\ j \\ k \end{bmatrix} = \begin{bmatrix} k \\ j \\ i \\ n \\ m \end{bmatrix}$$

↓

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} m \\ n \\ i \\ j \\ k \end{bmatrix} = \begin{bmatrix} m \\ j \\ n \\ i \\ k \end{bmatrix}$$

2. reversal: reversal of the ith loop is represented by the identity matrix. but with the ith diagonal element equal to −1 rather than 1.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} m \\ n \\ i \\ j \\ k \end{bmatrix} = \begin{bmatrix} m \\ -n \\ i \\ j \\ k \end{bmatrix}$$

3. skewing: skewing loop Ij by an integer factor f with respect to loop Ii maps iteration

$$(p_i, \dots p_{i-1}, p_i, p_{i+1}, \dots, p_{j-1}, p_j, p_{j+1}, \dots p_n)$$

to

$$(p_i, \dots p_{i-1}, p_i, p_{i+1}, \dots, p_{j-1}, p_j + f p_i, p_{j+1}, \dots p_n)$$

The transformation matrix T that produces skewing is the identity matrix, but the element $t_{j,i}$ equal to f rather than zero.

skew j by 2 with respect to n

$$\begin{array}{c} \\ m \\ n \\ i \\ j \\ k \end{array} \begin{array}{c} \begin{array}{ccccc} m & n & i & j & k \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{bmatrix} m \\ n \\ i \\ j \\ k \end{bmatrix} = \begin{bmatrix} m \\ n \\ i \\ 2n+j \\ k \end{bmatrix}$$

```
for m = 0 : 5
    for n = 0 : 7
        for i = 1 : 9
            for j = 1 : 7
                for k = 0 : 10
                    a[m, n, i, j, k] = a[m-1, n-1, i-1, j-1, k-1] * 3
```

```
for m = 0 : 5
    for n = 0 : 7
        for i = 1 : 9
            for j = n : 7+n
                for k = 0 : 10
                    a[m, n, i, j-n+2, k] =
                        a[m-1, n-1, i-1, j-n+1, k-1] * 3
```

Legality of loop transformation

We say that it is *legal* to apply a transformation to a loop nest if the transformed code can be executed sequentially, or in lexicographic order of the iteration space. We observe that if nodes in the transformed code are executed in lexicographic order, all data dependences are satisfied if the transformed dependence vectors are lexicographically positive. This obser-

"A unifying framework for iteration reordering transformations"

There is a mathmatical concept "unimodular transformation"

- What is unimodular framework, and what problems it solve?
  - Unimodular transformations is a unified framework that is able to describe any transformation that can be obtained by composing loop interchange, loop skewing, and loop reversal.
  - such a transformation is described by a unimodular linear mapping from the original iteration space to a new iteration space.

- Limitations of unimodular transformations.

> Unfortunately, unimodular transformations are limited in two ways: they can only be applied to perfectly nested loops and all statements in the loop nest are transformed in the same way. They therefore cannot represent some important transformations such as loop fusion, loop distribution and statement reordering.

> Our framework is designed to provide a uniform way to represent and reason about reordering transformations. The framework itself is not designed to decide which transformation should be applied. The framework should be used within some larger system, such as an interactive programming environment or an optimizing compiler. It is this surrounding system that is finally responsible for deciding which transformation should be applied. The framework does however provide some algorithms that would aid the surrounding system in its task. [KP94a] is an exam-

the same philosophy can be used for dl program.

→ This larger system could be a dl framework.

How dependences and mappings are represented?

The abstractions of data dependence directions and distances are sufficient for simple transformations such as unimodular transformations.