Notes on Dependence Analysis

Ayush Raina

March 14, 2025

Introductions

Why we need Dependence Analysis? We need to find best schedule which improves the locality and promotes parallelism. So we will first define some of the terms which are used in the context of dependence analysis.

Data Dependency

There is a data dependence from statement $S1$ to statement $S2$ if they both access the same memory location and atleast one of them stores into it. Secondly there should be a feasible run time execution path from $S1$ to $S2$.

Types Of Data Dependency

There are 3 main types of data dependencies:

- **Flow Dependence:** Also called True Dependence or Read After Write (RAW) Dependence. It occurs when S_1 writes a value and S_2 reads the value and is denoted by $(S_1\ \delta^F\ S_2)$.
- **Anti Dependence:** Also called Write After Read (WAR) Dependence. It occurs when S_1 reads a value and S_2 writes the value and is denoted by $(S_1\ \delta^A\ S_2)$.
- **Output Dependence:** Also called Write After Write (WAW) Dependence. It occurs when S_1 writes a value and S_2 writes the value and is denoted by $(S_1\ \delta^O\ S_2)$.

Loop Data Dependence

There exists a dependence from statement $S1$ to statement $S2$ in nested loops if and only if there exists two iteration vectors $\vec{i}$ and $\vec{j}$ such that:

- Either $\vec{i} < \vec{j}$ (Loop Carried Dependency) or $\vec{i} = \vec{j}$ and there exists a execution path from $S1$ to $S2$ in the body of the loop. (Loop Independent Dependency)
- S_1 accesses memory location M at iteration $\vec{i}$ and S_2 accesses memory location M at iteration $\vec{j}$.
- One of the accesses is a write.

Iteration Vectors and Iteration Spaces

We define iteration space as $I_s = \{\vec{i} \mid 0 \leq i_1 < N_1, 0 \leq i_2 < N_2, \ldots, 0 \leq i_n < N_n\}$. The iteration vectors are the points in the iteration space. The iteration vectors are denoted by $\vec{i} = (i_1, i_2, \ldots, i_n)$.

Lexicographic Ordering of Iteration Vectors

For $n = 2$, $(i, j) < (i', j')$ if $i < i'$ or $i = i'$ and $j < j'$. We can extend this notion to n dimensions as well. We can say $\vec{i} = (i_1, i_2, \ldots, i_n) < (i'_1, i'_2, \ldots, i'_n)$ if $\exists\ k\ s.t.\ i_k < i'_k$ and $i_j = i'_j\ \forall\ j < k$.

Some More Definitions

- **Dependence Distance Vector:** In a n-loop nest if S_1 is source of iteration $\vec{i}$ and S_2 is sink of iteration $\vec{j}$ then distance of this dependence denoted by $\vec{d} = \vec{j} - \vec{i}$. Each component of $\vec{d}$ is defined as $d_k = j_k - i_k$.
- **Dependence Direction Vector:** $<$ if $d_k \geq 0$, $>$ if $d_k \leq 0$ and $=$ if $d_k = 0$.

For example consider the following code block:

```
for(int i = 0;i < N; i++) {
  for(int j = 0;j < M; j++) {
    for(int k = 0;k < P; k++) {
    S1: A(i+1,j,k-1) = A(i,j,k) + 10
    }
  }
}
```

In this Case S_1 has flow dependence on itself. Distance vector is $(1, 0, -1)$ and direction vector is $(<, =, >)$. A dependence cannot exist until leftmost non zero component of distance vector is positive.

Dependency Tests

Consider the following code block:

```
  for(int i = 0;i < n; i++) {
    A[f(i)] = /* something */
    /* something */ = A[g(i)]
  }
```

Suppose there is a write operation in iteration $\vec{i_w}$ and read operation in iteration $\vec{i_r}$. Then this test is to detect any dependence from $\vec{i_w}$ to $\vec{i_r}$: $1 \leq \vec{i_w} \leq n, 1 \leq \vec{i_r} \leq n$ and $f(\vec{i_w}) = g(\vec{i_r})$.
For example consider the following code:

```
  for(int i = 1;i <= 100; i++) {
    for(int j = i;j <= 100; j++) {
      A[i,j+1] = A[i,j];
    }
  }
```

In this case let $\vec{i_w} = (i_w, j_w)$ and $\vec{i_r} = (i_r, j_r)$, then $f(\vec{i_w}) = (i_w, j_w + 1)$ and $g(\vec{i_r}) = (i_r, j_r)$. First we need to put contraints on induction variables which are as follows:

1. $1 \leq i_w \leq 100$
2. $i_w \leq j_w \leq 100$
3. $1 \leq i_r \leq 100$
4. $i_r \leq j_r \leq 100$

and $f(\vec{i_w}) = g(\vec{i_r})$ which implies $i_w = i_r$ and $j_w + 1 = j_r$. Are there integer solutions for these constraints ? Yes there are a lot of them:

$$\boxed{1 \leq i_w = i_r \leq j_w = (j_r - 1) \leq 99}$$

Conservative Testing

Subscript: This is a pair of expressions at same dimension. It has three different types:

- **ZIV** (Zero Induction Variable): $(2, 10)$
- **SIV** (Single Induction Variable): $A(i, i + 6)$
- **MIV** (Multiple Induction Variable): $A(i + j, j)$

Example: Consider the following code:

```
A[5,i+1,j] = A[10,i,k] + c;
```

Here we have 3 subscripts: $(5, 10), (i + 1, i), (j, k)$. The first subscript is ZIV, second is SIV and third is MIV.

Separable and Coupled Subscripts: Indices in separable subscripts do not occur in other subscripts. If two different subscripts contain same index they are coupled. Separable subscripts and coupled groups are handled separately. Consider the following example:

```
    A[i,j,j] = A[i,j,k] + 9;
```

In this case the second and third subscripts are coupled.

ZIV Test

Consider the array access as follows: $A[..., c_w, ...] = A[..., c_r, ...]$. If c_w and c_r are constants or loop invariant and $c_w \neq c_r$ then there is no dependence.
Consider the following code:

```
A[5, j+1, 10, k] = A[i, j, 12, k−1] + c;
```

We can see that the subscript $(10, 12)$ exists in 3rd dimension, is ZIV and $10 \neq 12$ so there is no dependence.

Strong SIV Test

Consider the array access as follows: $A[ai + c_w] = A[ai + c_r]$ with constrant: $ai + c_w = ai + c_r$. This problem has solution when $\boxed{i = \frac{c_w - c_r}{a}}$ is an integer and is in range of loop.
Consider the following code:

```
for(int i = 1; i <= 10; i++) {
  A[i+1] = A[i] + c;
}
```

This subscript is strong SIV as $a = 1, c_w = 1, c_r = 0$ and $i = \frac{1-0}{1} = 1 \in [1, 10]$. Hence there is a dependence.

General SIV Test or GCD Test

Consider the array access as follows: $A[a_w i + c_w] = A[a_r i + c_r]$ with constraint: $a_w i + c_w = a_r i + c_r$. Let $\gcd(a_w, a_r) = d$. If $d \nmid (c_w - c_r)$ then there is no dependence. Else there are many possible solutions.
For example consider the following code:

```
for(int i = 0;i < 10; i++) {
  A[2i+1] = A[4i] + c;
}
```

This subscript is a strong SIV as $a_w = 2, a_r = 4, c_w = 1, c_r = 0$ and $d = \gcd(2, 4) = 2$ and $2 \nmid (1 - 0)$ so there is no dependence.

Banerjee's Test

Consider the array access of the form $A[a_w i + c_w] = A[a_r i + c_r]$ with constraint: $a_w i + c_w = a_r i + c_r$. So satisfy this constraint let $h(i_w, i_r) = a_w i_w - a_r i_r + c_w - c_r$. If $max(h) \geq 0$ and $min(h) \leq 0$ then by intermediate value theorem there is a solution.
For example consider the following code:

```
for(int i = 1;i <= 100; i++) {
  A[2i+3] = A[i+7];
}
```

In this case $h(i_w, i_r) = 2i_w - i_r + 3 - 7$, hence $\boxed{h(i_w, i_r) = 2i_w - i_r - 4}$, now we need to find the maximum and minimum value of $h(i_w, i_r)$. We can easily see $max(h) = 2*100-1-4 = 195$ and $min(h) = 2*1-100-4 = -102$. Hence there is a dependence.

Omega Test

Locality Analysis

Recap

Iterations are executed in lexicographic order of their iteration vectors. If $\vec{i_2} \succ \vec{i_1}$ then iteration $\vec{i_2}$ is executed after $\vec{i_1}$.

Unimodular Transformations

The elementary permutation matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ performs the loop interchange transformation to iteration space containing two loops as follows: $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} i_2 \\ i_1 \end{bmatrix}$. Since matrix transformation T is a linear transformation on the iteration space, which means $T\vec{i_2} - T\vec{i_1} = T(\vec{i_2} - \vec{i_1})$.Therefore if $\vec{d}$ is a distance vector in original iteration space then $T\vec{d}$ is a distance vector in transformed iteration space. Loop Interchange is valid if $T\vec{d} \succ 0$.

Unimodular Matrices: Square matrices with integral components with determinant equal to one or negative
Theorem: Let D be the set of all distance vectors of a loop nest. A unimodular transformation T is legal iff $\forall\ \vec{d} \in D : T\vec{d} \succ 0$