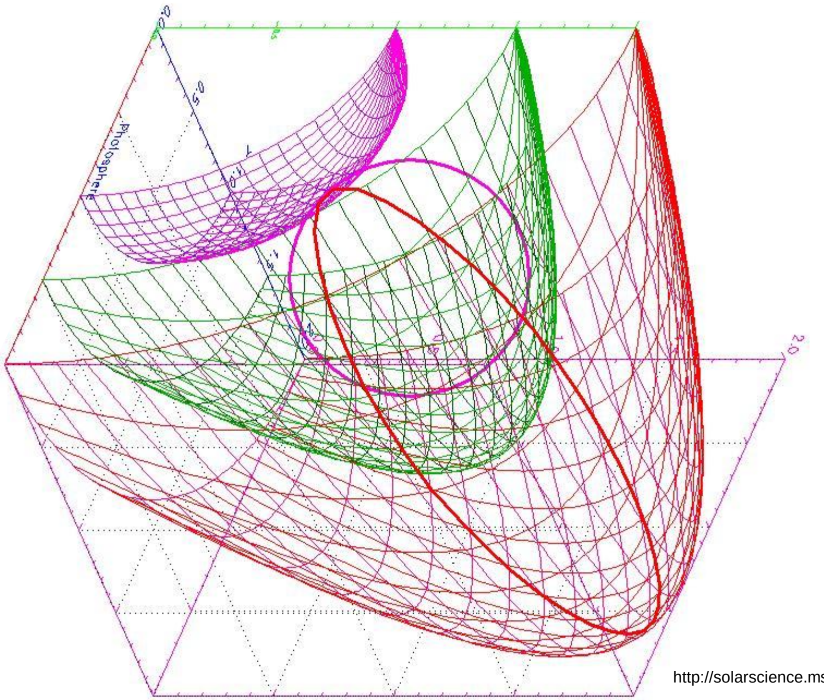
cse5441 - parallel computing

loop dependence analysis



http://solarscience.msfc.nasa.gov/research/3d_fields/StretchviaBezierCurves4.jpg

statement independence

Why might we want to re-order program statements?

- better cache performance
- parallel programs

What determines whether program statements can be re-ordered?

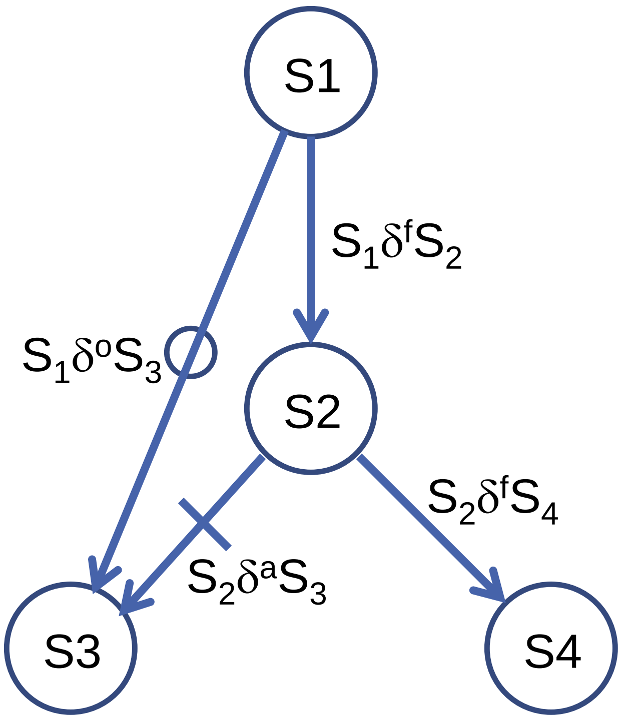
- dependence on prior computations

data dependencies

serial code segment:

```
S1:   a = b + c
S2:   d = a * 2
S3:   a = c + 2
S4:   e = d + c + 2
```

stmt	read	write	
S1	b,c	a	
S2	**a**	d	$S_1\delta^f S_2$
S3	c	**a**	$S_1\delta^o S_3$, $S_2\delta^a S_3$
S4	**d**,c	e	$S_2\delta^f S_4$



NOTE:
a non-reorderable dependence is created when:

- multiple accesses to same memory address
- one or more of these accesses is a write

data dependence types

flow-dependence: occurs when a variable which is **assigned a value** in one statement, e.g. S_1, is **read by another** statement, e.g. S_2, later. Written as $S_1\delta^f S_2$.

anti-dependence: occurs when a variable **read** by one statement, e.g. S_1, is assigned an **updated value in another** statement, e.g. S_2, later. Written as $S_1\delta^a S_2$.

output-dependence: occurs when a variable which is **assigned a value** in one statement, e.g. S_1, is **later re-assigned** an updated value in another statement, e.g. S_2, later. Written as $S_1\delta^o S_2$.

data dependencies

S1: a = b + c

S2: d = a * 2

S3: a = c + 2

S4: e = d + c + 2

IN(S_i): the set of memory locations read by the statement S_i.

OUT(S_i): the set of memory locations written by the statement S_i.

(note a specific memory location may be both IN(S_i) and OUT(S_i).)

stmt	read	write	
S1	b,c	a	
S2	a	d	$OUT(S1) \cap IN(S2) = \{a\} \neq \varnothing \rightarrow S_1\delta^f S_2$
S3	c	a	$OUT(S1) \cap OUT(S3) = \{a\} \neq \varnothing \rightarrow S_1\delta^o S_3$,
			$IN(S2) \cap OUT(S3) = \{a\} \neq \varnothing \rightarrow S_2\delta^a S_3$
S4	d,c	e	$OUT(S2) \cap IN(S4) = \{d\} \neq \varnothing \rightarrow S_2\delta^f S_4$

NOTE: we are working at the statement level, there is no dependence within a statement

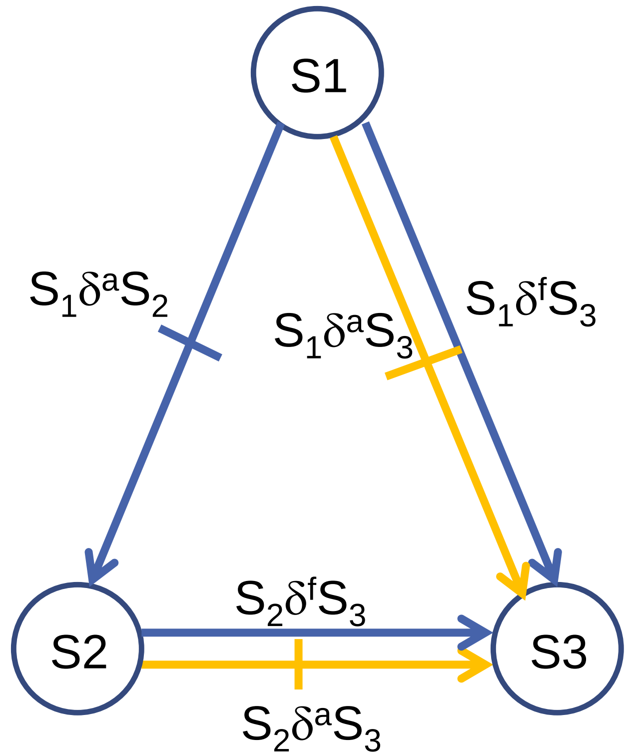
it's your turn . . .

S1: a = a + b + c

S2: b = b + c + d

S3: c = a + b

stmt	read	write	
S1	a,b,c	a	
S2	b,c,d	b	$S_1\delta^aS_2$
S3	a,b	c	$S_1\delta^fS_3$ $S_1\delta^aS_3$ $S_2\delta^fS_3$ $S_2\delta^aS_3$



loop data dependencies

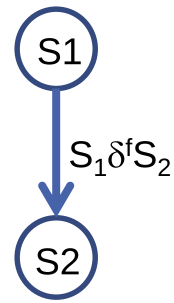
```
        for (int i = 1; i <= 50; i++)
S1:       A( i )  =  B( i-1)  + C( i )
S2:       B( i )  =  A( i+2) + C( i )
```

```
S1(1):    A(1)  =  B(0) + C(1)
S2(1):    B(1)  =  A(3) + C(1)

S1(2):    A(2)  =  B(1) + C(2)
S2(2):    B(2)  =  A(4) + C(2)

S1(3):    A(3)  =  B(2) + C(3)
S2(3):    B(3)  =  A(5) + C(3)

S1(4):    A(4)  =  B(3) + C(4)
S2(4):    B(4)  =  A(6) + C(4)
  .
  .
  .

S1(50):   A(50)  =  B(49) + C(50)
S2(50):   B(50)  =  A(52) + C(50)
```

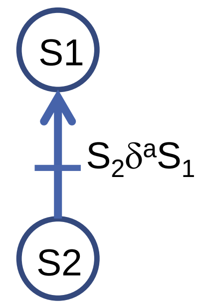
<u>NOTE:</u>
we ignore boundary conditions

loop data dependencies

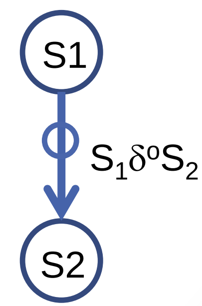
```
A       for (int i = 1; i <= 50; i++)
S1:       A( i+n )  =  whatever(happens, next)
S2:       B( i )    =  A( i )
```




```
B       for (int i = 1; i <= 50; i++)
S1:       A( i-n )  =  whatever(happens, next)
S2:       B( i )    =  A( i )
```




```
C       for (int i = 1; i <= 50; i++)
S1:       A( i )    =  whatever(happens, next)
S2:       A( i )    =  this(not, that)
```

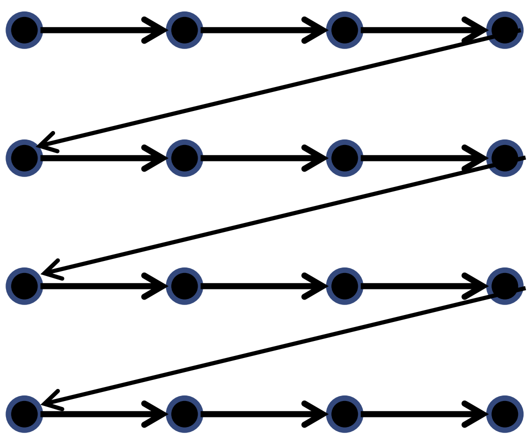


notice: all cases – multiple accesses to same memory location!

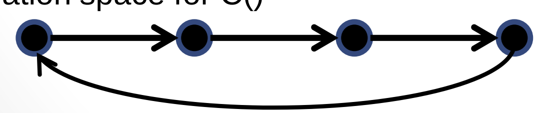
iteration space

```
for (int i = 0; i <= 4; i++)
    for (int j = 0; j <= 4; j++)
        A( i, j ) = B( i, j ) • C( j )
```

iteration space for A() and B()

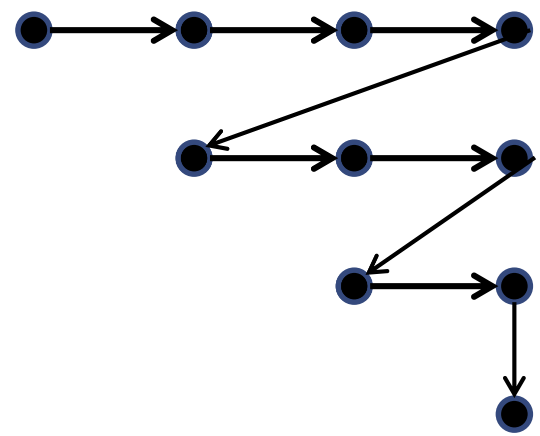


iteration space for C()



```
for (int i = 0; i <= 4; i++)
    for (int j = i; j <= 4; j++)
        A( i, j ) = B( i, j ) • C( j )
```

iteration space for A() and B()



iteration space for C()

J. S. Jones

loop exit state

```
for (int i = 0; i <= 4; i++)
    for (int j = 0; j <= 4; j++)

        A( i, j ) = B( i, j ) op C( j )
```
where *op* is an arbitrary operator

next statement(A, B, C)

A = { 0,0,0,0,
1,2,3,4,
5,5,5,5,
6,7,8,9 }

B = { 1,3,5,7,
1,2,3,4,
9,9,9,9,
6,7,8,9 }

C = { 0,0,0,0,
4,4,4,4,
6,6,6,6,
6,7,8,9 }

```
for (int j = 0; j <= 4; i++)
    for (int i = 0; i <= 4; j++)

        A( i, j ) = B( i, j ) op C( j )
```
where *op* is an arbitrary operator

next statement(A, B, C)

A = { same? }

B = { same? }

C = { same? }}

lexicographic ordering

aka *lexical* or *alphabetical* or *natural* ordering

given two length-2 index vectors: $\mathbf{I} = (i_1, i_2)$ and $\mathbf{J} = (j_1, j_2)$

$(i_1, i_2) \prec (j_1, j_2) \leftrightarrow (i_1 < j_1)$ or $((i_1 = j_1)$ and $(i_2 < j_2))$

$(0, 1) \prec (1, 0)$
$(0, 0) \prec (0, 2)$

given two arbitrary length index vectors:
$\mathbf{V_1} = (i_1, \ldots i_m, i_{m+1}, \ldots i_n)$ and $\mathbf{V_2} = (j_1, \ldots j_m, j_{m+1}, \ldots j_n)$

$V_1 \prec V_2 \leftrightarrow (i_1 < j_1)$ or $((i_1 - i_m = j_1 - j_m)$ and $(i_{m+1} < j_{m+1}))$

$(0, 4, 4, 4, 4, 8) \prec (1, 1, 1, 1, 1, 0)$
$(2, 3, 4, 5, 6, 7) \prec (2, 3, 4, 5, 7, 7)$

J. S. Jones

loop nest dependencies

consider loops of the form:

---

for (int $i_1 = L_1$; $i_1 <= U_1$; i_1++)
    for (int $i_2 = L_2$; $i_2 <= U_2$; i_2++)
    . . .
        for (int $i_n = L_n$; $i_n <= U_n$; i_n++)

            BODY(i_1, i_2, … i_n)

given: iterations $\mathbf{I^v} = (i_1, i_2, \dots i_n)$ and $\mathbf{J^v} = (j_1, j_2, \dots j_n)$
memory location **M**

---

there exists one or more loop dependencies if:

- $\mathbf{I^v} \prec \mathbf{J^v}$
- both BODY($\mathbf{I^v}$) and BODY($\mathbf{J^v}$) reference **M**
- at least one such reference is a write

J. S. Jones

distance vectors

given: - iterations $\mathbf{I}^v = (i_1, i_2, \ldots i_n)$ and $\mathbf{J}^v = (j_1, j_2, \ldots j_n)$

- a dependence from BODY($\mathbf{I}^v$) to BODY($\mathbf{J}^v$)

the dependence distance vector

$$\Delta^v = (d_1, d_2, \ldots d_n)$$

where $\forall d_\alpha : d_\alpha = j_\alpha - i_\alpha$

direction vectors

given: - iterations $\mathbf{I}^v = (i_1, i_2, \ldots i_n)$ and $\mathbf{J}^v = (j_1, j_2, \ldots j_n)$

- a dependence from BODY($\mathbf{I}^v$) to BODY($\mathbf{J}^v$)

- distance vector $\Delta^v = (d_1, d_2, \ldots d_n)$

- the sign function $\Sigma(x_\alpha) = \begin{cases} - & \text{if } x_\alpha < 0 \\ 0 & \text{if } x_\alpha = 0 \\ + & \text{if } x_\alpha > 0 \end{cases}$

the dependence direction vector

$$\delta^v = (\, \Sigma(d_1), \Sigma(d_2), \ldots \Sigma(d_n)\,)$$

dependence vectors

example

```
for (int i = 1;  i <= 5;  i++)
   for (int j= 1;  j <= 5;  j++)
      A( i, j )  =  A( i, j-3)  +  A( i-2, j)  +  A( i-1, j+2)  +  A( i+1, j-1)
```

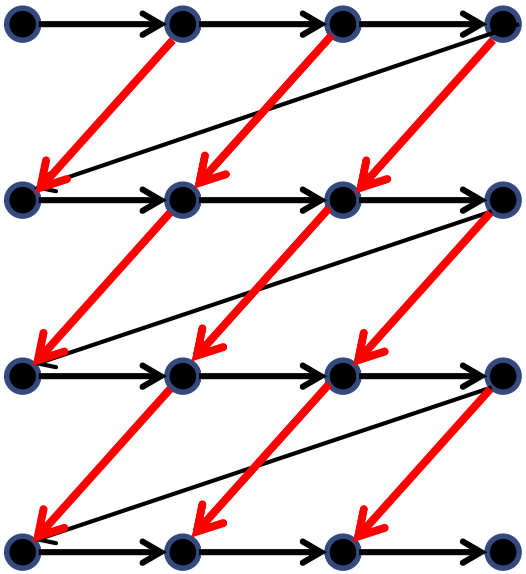
RHS ref.	type	Δ distance vector	δ direction vector
A(i, j-3)	flow	(0, 3)	(0, +)
A(i-2, j)	flow	(2, 0)	(+, 0)
A(i-1, j+2)	flow	(1, -2)	(+, -)
A(i+1, j-1)	anti	(-1, 1)	(-, +)

"step" from the write iteration (LHS)
to
the read iteration (RHS)
for
distance and direction

loop interchange

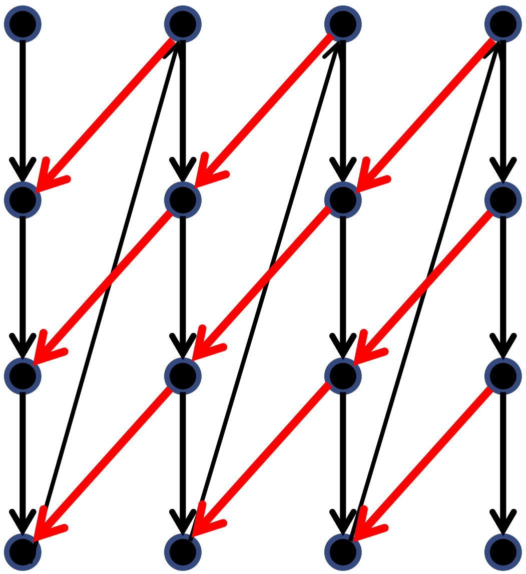
A

```
for (int i = 0; i <= n; i++)
    for (int j = 0; j <= n; j++)
        A( i, j) = A( i-1, j+1) * .9
```



B

```
for (int j = 0; j <= n; j++)
    for (int i = 0; i <= n; i++)
        A( i, j) = A( i-1, j+1) * .9
```

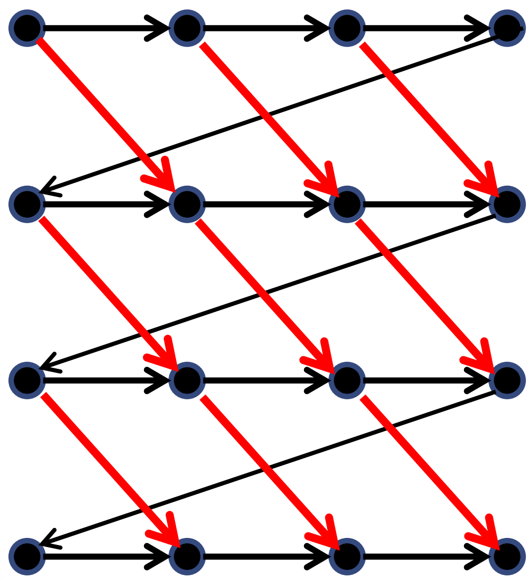


loop interchange

example 2

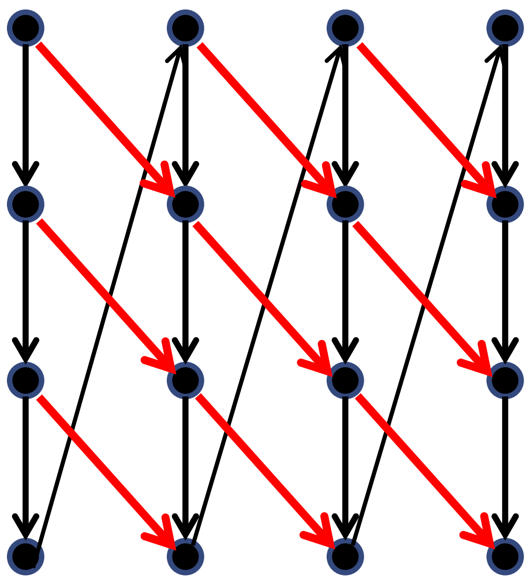
A

```
for (int i = 0; i <= n; i++)
    for (int j = 0; j <= n; j++)
        A( i, j) = A( i-1, j-1) * .9
```



B

```
for (int j = 0; j <= n; j++)
    for (int i = 0; i <= n; i++)
        A( i, j) = A( i-1, j-1) * .9
```



OSU CSE 5441

J. S. Jones

dependence vectors

example 3

```
for ( i = 0;  i < MAX;  i++ )
    for ( j = 0;  j < MAX;  j++ )
        for ( k = 0; k < MAX; k++ )
            A[ i, j, k]  +=  A[ i-1, j-1, k+1 ]
```

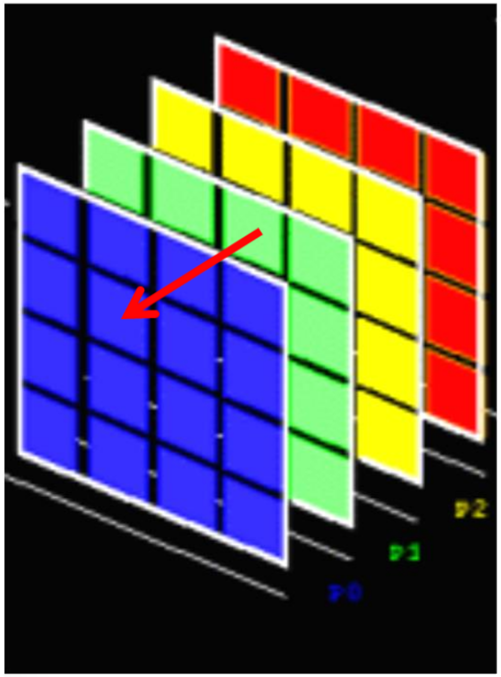
RHS ref.	type	distance vector	direction vector
A [i-1, j-1, k+1]	flow	(1, 1, -1)	(+, +, -)
		(i, j, k)	(i, j, k)

permitted permutations: IJK, IKJ, JKI, JIK

loop interchange

example 3

I J K



```
for ( i = 0;  i < MAX;  i++ )
     for ( j = 0;  j < MAX;  j++ )
          for ( k = 0; k < MAX; k++ )
               A[ i, j, k]  +=  A[ i-1, j-1, k+1 ]
```

loop nest	direction vector	permissible re-orderings
I J K	(+ + -)	yes
J I K	(+ + -)	yes
I K J	(+ - +)	yes
J K I	(+ - +)	yes
K I J	(- + +)	no
K J I	(- + +)	no

http://math.nist.gov/jazznet/layout.gif

it's your turn . . .

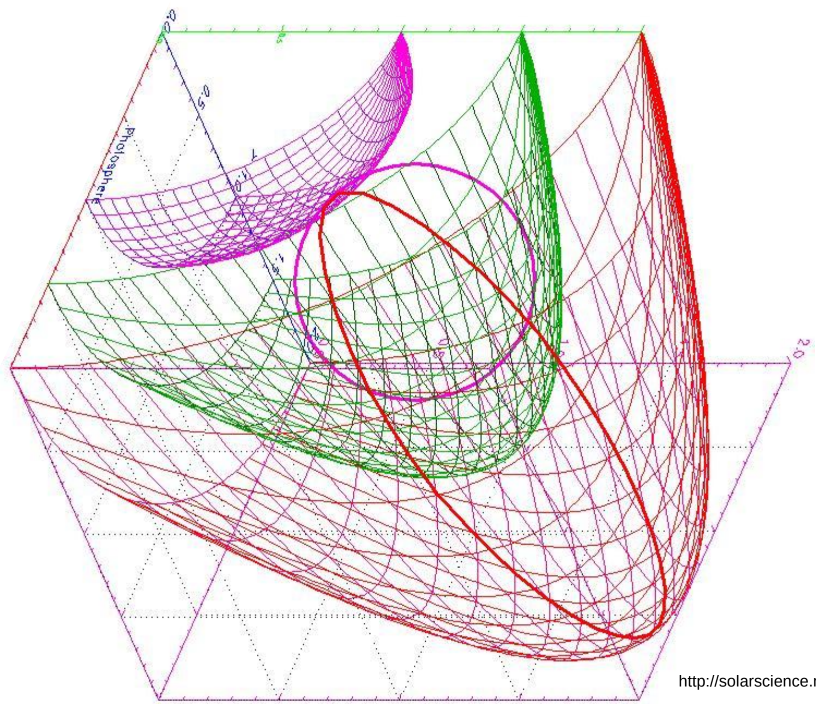
example 4

```
for ( i = 0;  i < MAX;  i++ )
    for ( j = 0;  j < MAX;  j++ )
        for ( k = 0; k < MAX; k++ )
            for ( m = 0; m < MAX; m++ )
                BODY(i, j, k, m)
```

	BODY	distance vector	direction vector
a.)	R[m] += A[i, j, k]	---	---
b.)	R[m] += A[i, j, k, m] * A[i-1, j-1, k-1, m-1]	---	---
c.)	A[i, j, k, m] += A[i-1, j, k, m]	(1, 0, 0, 0)	(+, 0, 0, 0)
d.)	A[i, j, k, m] += A[i-1, j-1, k, m]	(1, 1, 0, 0)	(+, +, 0, 0)
e.)	A[i, j, k, m] += A[i-1, j-1, k+1, m+1]	(1, 1, -1, -1)	(+, +, -, -)

cse5441 - parallel computing

loop dependence analysis



http://solarscience.msfc.nasa.gov/research/3d_fields/StretchviaBezierCurves4.jpg

dependence analysis - answers

slide 20:

a.) no dependencies, fully re-orderable
(no instances where same address is read/written

b.) no dependencies, fully re-orderable
(no instances where same address is read/written

c.) flow dependence, fully re-orderable
(sign of vector cannot change)

d.) flow dependence, fully re-orderable
(sign of vector cannot change)

e.) flow dependence, partially re-orderable
valid from IJKM: IJMK, IKJM, IKMJ, IMJK, IMKJ,
JIKM, JIMK, JKIM, JKMI, JMIK, JMKI